



University
of Glasgow

Song, Yulun (2015) *An authoring and presentation environment for interactive worked examples*. PhD thesis

<http://theses.gla.ac.uk/6152/>

Copyright and moral rights for this thesis are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the Author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the Author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given



University
of Glasgow

**An Authoring and Presentation
Environment for Interactive
Worked Examples**

Yulun Song

Submitted for the Degree of

Doctor of Philosophy

School of Computing Science

College of Science and Engineering

University of Glasgow

February 2015

Abstract

This dissertation describes an authoring environment, called IWE, which allows a teacher to develop computer-based interactive worked examples without bespoke programming. The focus is on worked examples that involve transforming one representation into another using judgments not algorithms or rules. The worked examples created are all drawn from Computing Science; for example, transforming a requirements specification into an entity-relationship diagram. Teachers model the problem-solving process as a sequence of steps demonstrating how the problem is translated step-by-step into a solution, explaining the decision-making in each step. They can incorporate questions within the examples to increase student engagement and encourage students to do active thinking. Students interact with the transformation process at their own pace to obtain experience of problem-solving. Teachers are able to evolve the examples based on feedback from students and usage data from the system.

A review of educational literature identified the best practice guidelines for designing and presenting effective worked examples for novices and faded worked examples for intermediate learners. These guidelines informed the essential requirements of IWE. A prototype authoring environment was designed, implemented and evaluated. Educational literature also recommends using worked examples combined with practice of problem solving. A field study was conducted applying these recommendations to evaluate the usability of IWE. Evaluations were carried out with teachers to assess their ability to create and modify interactive worked examples while the teaching of their courses was in progress. Evaluations were also carried out with students to assess the usability of IWE.

The main conclusion of this research, based on analysis of the evaluations, is that the prototype of IWE is useable by both teachers and students. It allows teachers to create interactive worked examples following best practice and evolve existing examples on the basis of feedback. It allows students to use interactive worked examples independently following best practice. Finally, the dissertation identifies some possibilities for widening the scope of this research.

Table of Contents

Abstract	i
Table of Contents	ii
Acknowledgement	v
Author’s Declaration	vii
Chapter 1 Introduction	- 1 -
1.1 Motivation	- 1 -
1.2 The Scope of this Research	- 3 -
1.3 The Thesis Statement and Research Questions	- 6 -
1.4 Contribution of the Research	- 8 -
1.5 The Structure of Thesis	- 9 -
Chapter 2 Education Background	- 11 -
2.1 Educational Context for the Research	- 11 -
2.1.1 Transformation-based Problems.....	- 11 -
2.1.2 Solving Transformation-based Problems Requires Judgement	- 12 -
2.1.3 Judgements Require Prior Successful Experiences	- 19 -
2.2 Understanding Learning at the Cognitive Level	- 21 -
2.2.1 Human Cognitive Architecture and Schema Development	- 21 -
2.2.2 Cognitive Load, Cognitive Overload, and Learning	- 25 -
2.3 Cognitive Apprenticeship to Develop Schemata	- 26 -
2.4 Current Instructional Designs for Transformation-based Problems	- 28 -
2.4.1 Weaknesses in the Current Instructional Design	- 30 -
2.5 Worked Example Research	- 36 -
2.5.1 Worked Examples Effect	- 36 -
2.5.2 Faded Worked Examples	- 39 -
2.5.3 Guidance for Design Worked Example and Faded Worked Example	- 41 -
2.6 Need for Interactive Worked Examples	- 44 -
2.7 Challenges of Interactive Worked Examples	- 47 -
2.7.1 Productivity and Customization in Learning Objects	- 47 -
2.7.2 TPACK Framework	- 49 -
2.8 Authoring Environments and Tools	- 52 -
2.8.1 Empowering the Teacher	- 52 -
2.8.2 The Essential Features of the Teacher User Interface	- 54 -
2.8.3 The Essential Features of the Student User Interface.....	- 55 -
2.9 Summary of the Research Context	- 56 -
2.10 The Proposed Research Work	- 57 -
2.10.1 The Concept of IWE.....	- 57 -
2.10.2 The IWE Model	- 59 -
2.10.3 IWE Requirements.....	- 61 -
Chapter 3 Usability Literature Review	- 63 -
3.1 Usability Definition	- 63 -
3.2 General Methods of Usability Evaluation	- 64 -
3.3 Extra Work for the Usability Evaluation of Educational Software	- 68 -
Chapter 4 Requirements	- 73 -
4.1 General Requirements	- 73 -
4.1.1 The Overall Structure of IWE.....	- 74 -
4.1.2 Basic Concepts of IWE	- 76 -
4.1.3 The IWE Data Model	- 77 -
4.2 Detailed Requirements	- 78 -

4.3 Use Case Descriptions	- 79 -
4.3.1 Teacher’s Use Case Descriptions	- 80 -
4.3.2 Student’s Use Case Descriptions	- 86 -
4.4 Other Requirements	- 88 -
4.4.1 System Requirements	- 88 -
4.4.2 Log Requirements	- 89 -
4.4.3 Help Function	- 89 -
Chapter 5 Design and Implementation	- 90 -
5.1 The Implementation and Deployment Architecture of IWE	- 90 -
5.1.1 The Implementation of IWE	- 90 -
5.1.2 The Deployment of IWE	- 91 -
5.2 Initial User Interface Designs	- 94 -
5.2.1 Teacher User Interface Design	- 94 -
5.2.2 Student User Interface Design	- 103 -
5.2.3 Feedback Mechanism within IWE Design	- 106 -
5.3 Final User Interface Implementation	- 108 -
5.3.1 Teacher’s User Interfaces Implementation	- 108 -
5.3.2 Student’s User Interfaces Implementation	- 116 -
5.3.3 System Help User Interface	- 121 -
5.4 Conclusion	- 121 -
Chapter 6 Evaluations	- 122 -
6.1 Timeline of Evaluations	- 122 -
6.2 Initial Evaluations for Testing Purposes	- 125 -
6.2.1 Evaluation of Prototype Student User Interface	- 125 -
6.2.2 Survey of Learning Needs.....	- 126 -
6.2.3 Follow-up Initial Feasibility Study.....	- 128 -
6.2.4 Pilot Test of Teacher User Interface.....	- 130 -
6.3 Teacher Evaluation	- 132 -
6.3.1 Overview the Process of Teacher Evaluation	- 132 -
6.3.2 Built Examples	- 133 -
6.3.3 Experience of Building Examples.....	- 135 -
6.3.4 Conclusions.....	- 136 -
6.4 Level-1 Volunteer Students Survey Using IWE	- 137 -
6.4.1 Motivation.....	- 137 -
6.4.2 Overview of the Design of the Survey.....	- 137 -
6.4.3 Results of IWE Using Survey.....	- 138 -
6.5 HCI Expert Evaluation	- 146 -
6.5.1 Suggestions Based on the Use of the Viewing Controls.....	- 147 -
6.5.2 Suggestions Related to Ask Questions	- 147 -
6.5.3 Suggestions for Future Improvement Work.....	- 148 -
6.6 Follow-up One-to-one Usability Investigation	- 148 -
6.6.1 Overview	- 148 -
6.6.2 Usability Investigation Questions.....	- 149 -
6.6.3 Usability Investigation Process.....	- 150 -
6.6.4 Results of Usability Investigation	- 152 -
6.7 Conclusions	- 161 -
6.7.1 Enhancing the User Interfaces	- 163 -
6.7.2 Final Evaluation of an Improved Version of IWE.....	- 167 -
Chapter 7 Field Study of IWE	- 168 -
7.1 The IWE-based Interventions	- 169 -
7.1.1 Educational Context for Using IWE	- 169 -
7.1.2 The Use of IWE in CS1CT	- 170 -
7.1.3 Planned Use in Lab Exercises	- 171 -

7.1.4 Deepening Conceptual Understanding	- 173 -
7.1.5 Practice with Exam Question Formats	- 174 -
7.2 Users Identification for the IWE Evaluation	- 176 -
7.3 Usability of IWE	- 180 -
7.3.1 Effectiveness of Short-term Improvements	- 180 -
7.3.2 SUS Questionnaire Results	- 180 -
7.4 Utility of IWE	- 182 -
7.4.1 A Conceptual Discussion	- 182 -
7.4.2 Possible Reasons for Giving IWE a Low Score in the SUS Questionnaire.....	- 194 -
7.5 Teacher’s Evaluation Result	- 198 -
7.5.1 Evolution of the <i>HorizontalLinesProgram</i>	- 198 -
7.5.2 Lessons Learned From <i>FindErrorExample</i>	- 203 -
7.6 Summary.....	- 203 -
Chapter 8 Discussion and Future Work.....	- 206 -
8.1 Verifying the Thesis Statement	- 207 -
8.2 Limitations	- 217 -
8.3 Suggested Structure of Future Work.....	- 218 -
8.3.1 Analysis of Possible Future Work	- 219 -
8.3.2 Possible Further Research	- 220 -
Bibliography	- 226 -
Appendices.....	- 233 -
Appendix 1: Participant Consent Form: IWE Usability Survey.....	- 233 -
Appendix 2: IWE Evaluation Questionnaire	- 234 -
Appendix 3: Results of IWE Evaluation	- 235 -
Appendix 4: One-to-one Usability Investigation Questions	- 239 -
Appendix 5: Result of One-to-one Usability Investigation Survey.....	- 240 -
Appendix 6: Interview Results of One-to-one Usability Investigation	- 244 -
Appendix 7: Teacher’s Evaluation Report	- 254 -
Appendix 8 Lab Exercise Sheet for Week 7	- 265 -
Appendix 9 Student Lab Exam Question.....	- 268 -
Appendix 10 Related Questions in Student Written Exam.....	- 271 -
Appendix 11 The Implementation Structure of IWE.....	- 273 -
Appendix 12 Extensible Markup Language (XML) Technology	- 275 -
Appendix 13 System Usability Scale (SUS) Questionnaire and its Benchmark Lookup Table	- 278 -

Acknowledgement

My research work would not have been finished without the help, support and encouragement from the following people.

Firstly, I would like to express my sincerest gratitude to my three first supervisors in turn, Dr. Richard Cooper, Prof. Ray Welland and Prof. Quintin Cutts, for all their great knowledge, guidance, patience, and enormous support throughout my doctoral work. Dr. Richard Cooper offered me an opportunity to become a doctoral researcher, however, due to family reason he retired earlier and transferred me to Prof Ray Welland. Prof. Ray Welland directed and provided scaffolding me on the right track for doing research. Even after retirement, his continuous support and advice deeply inspired me, not only in finishing the research, but also deeply influencing me on the way of doing things right. Finally, Prof. Quintin Cutts took over and his solid research experience in computing education, helpful advice, patient guidance, enlightening discussions and valuable feedback, helped me to finish this valuable research. Without their enthusiastic support and continuous encouragement, I would have never been able to stand at this point in my life.

Secondly, I would like to thank my second supervisor Dr. Helen C. Purchase for her critical suggestions, useful discussions and fruitful experience of doing the evaluation in computing science subject.

Third, I would like to thank Dr. Marilyn R McGee-Lennon for the contribution in the teacher's evaluation of this authoring system. Mr. Philip Gray, an excellent HCI and software engineering expert, gave useful suggestions for the usability evaluation of IWE.

Fourth, I would like to express my deeply gratitude to all my family for their endless love. In particular, my father Min Song, my mother GuiYun Zhao, my father in law HuanLiang Wu and my mother in law LanYing Wang for encouraging and supporting me to achieve one of my dreams. Great thanks should also give to my wise and lovely wife XiaoHua Wu for her studying with me in Scotland, as we spent our happiness time together. Thanks for her patience, understanding,

huge encouragement, and brightening up my life. Millions of thanks give to her bringing my lovely daughter Qiancheng Song.

Thanks also give to Mr. Stewart Macneill and Mr. Douglas Macfarlane, offered technical support in setting and distributing IWE in the Level-1 student's lab of the School of Computing Science in University of Glasgow.

I would also like to thank the people in my School, providing help and suggestions on debugging the code during the process of development. Not only for their friendship, enjoyable times, but also for exchanging of knowledge during my years of study and research. I would like to say thanks to all the students who took the evaluations of different versions of IWE during years. Thanks to their volunteer work and useful feedback on different aspects of the system for improving IWE.

Author's Declaration

I declare that the work presented in this thesis is my own unless otherwise stated by reference, and that the work has not been submitted for any other degree or professional qualification except as specified.

Signed:

Date:

Chapter 1 Introduction

This dissertation concerns computer science education and applied computer science. It identifies a particular challenging learning and teaching area within computer science concerning the development of problem-solving skills; it presents the motivation, design and evaluation of a technology-based tool to support students' development of these essential skills.

1.1 Motivation

The motivation for this research initially came from a teacher requiring better ways of creating and presenting worked examples for teaching a database course. Teaching and learning database concepts is hard, for example designing an ER diagram to represent a customer problem. It requires the teacher to clearly explain the high-level abstract modelling concepts involved in ER diagrams first. Second, students need to see enough examples to understand how the modelling concepts are used. Third, it requires the students to map a real-world problem into a model, built out of the ER concepts, which can be used to implement a database application. Another difficulty in teaching databases is that modern database systems provide a high-level user interface, which hides much of the implementation detail. So it is difficult to give the students practical experience of the intuitions and techniques involved[1], for example, using the Aqua Data Studio [2] in creating SQL code from ER diagrams.

Problems in teaching computer science are not only in the database area, but also in other topics. Similar problems occur in programming courses. For example, the teacher can clearly explain the concepts and the students can understand and self-explain the concepts. However, when students work on their own to produce solutions to a target problem, some cannot get started or else they produce widely-varying solutions, some of which do not work.

The common factor in these areas is the nature of the problem solving process. Deciding on the solution path is a heuristic process - there is no simple rule or algorithmic process to follow. Even when students do manage to produce working solutions they can be very good, good, or just adequate. Pólya understood this well in his book on mathematical problem solving[3]. He offered

no such simple set of rules, instead he gave over 200 pages of short essays designed to give insight into the process. Finding good solutions is based on previous experience. Students, especially novices, lack the experience to make judgments to produce a good solution. Hence, novices need support on how to make judgments and demonstrations of how to generate a good solution. Other approaches are possible, for example the use of programming patterns, e.g. as exemplified in the "Gang of Four" book [4], but even then, students will need support in understanding how and when to use patterns.

Research into worked examples has shown the value of using them in education, as the step-by-step demonstration can help novices to develop experience for supporting problem-solving later on. Best practice models of creating and using worked examples are also available to teachers, which are well explained in [5], [6] and [7]. However, current instructional designs typically used in Computing Science do not permit adequate exposure to worked examples and therefore students are required to attempt free problem solving before they have developed the necessary expertise.

Technology can support the transformation of worked examples into an interactive format, as computer-based courseware, with extra benefits. For example, an instructor can design as many steps as needed; important contents can be highlighted within a step; a question step can be added in with the aim of encouraging students to think about what will happen in the next step; and moving through the steps of the worked example is under students' control. Hence, interactive worked examples are able to be used by novices to develop the initial experience of solving problems, shown for example in ADbC [8].

Software used in educational courses (courseware) designed to be used by students to directly support their self-study activities after initial exposition by a teacher is typically known as secondary courseware [9]. Enthusiastic teachers want to be able to create or modify the content of secondary courseware in order to embed it within their own context according to their own pedagogical design. However, designing and delivering effective secondary courseware is a very complex and costly task which requires experts from several areas working together most of the time, e.g. experts in content, pedagogy and technology. Occasionally a teacher has enough time and programming expertise combined

with the necessary pedagogical and subject knowledge to build their own secondary courseware but this is rare. Therefore, the majority of teachers have to rely on paying experts to develop software to their specification or, more likely, buy off-the-shelf ('shrink-wrapped') secondary courseware.

The dynamic nature of teaching, where for example student misunderstandings or shortcomings in a teacher's materials only surface after the course has started, requires teachers either to adjust existing secondary courseware or offer new secondary courseware in a very short time. Any modification to existing courseware is problematic because in most instances this requires some level of editing, re-contextualization or re-purposing of existing resources for them to be useful. The fact that most off-the-shelf or shrink-wrapped secondary courseware cannot be easily adapted, either because of licensing restrictions or because it has been created or packaged using tools requiring a high level of technical expertise, limits its pedagogical value. In particular, the vast majority of teachers do not have enough technical expertise to make changes to this type of software.

Hence, there is a need to provide teachers with the means to create and evolve their secondary courseware to fit the contextualised nature of teaching and learning. Reducing the level of technical expertise required for teachers to work on secondary courseware is a key step in meeting this need. It is a worthwhile challenge for computing scientists to work on.

1.2 The Scope of this Research

The education background in Chapter 2 argues for the effectiveness of worked examples for acquiring and developing the initial experience of problem-solving, based on a broad range of existing research. The educational value of worked examples is highlighted, and best practice principles for constructing and effectively using worked examples are identified.

Based upon this solid foundation, this dissertation focuses on the development of an interactive worked examples authoring tool, known as IWE, and a thorough evaluation of the tool with respect to teachers being able to create and modify, and students study from, worked examples following best-practice principles.

The dissertation does not aim to assess the educational value of worked examples themselves, viewing this as work already completed.

The scope of this research focusses on delivering worked examples in computing science of the kind described earlier, not only because the researcher is within and familiar with this subject, but also because there are a wealth of challenging teaching problems in this area. The non-algorithmic process of solving this particular type of problem requires students to apply different guidelines in different steps and make judgements based on prior experience, rather than simply applying algorithms.

The highlighted area in Figure 1.1 indicates the educational context of this research, the detailed explanation of this figure follows.

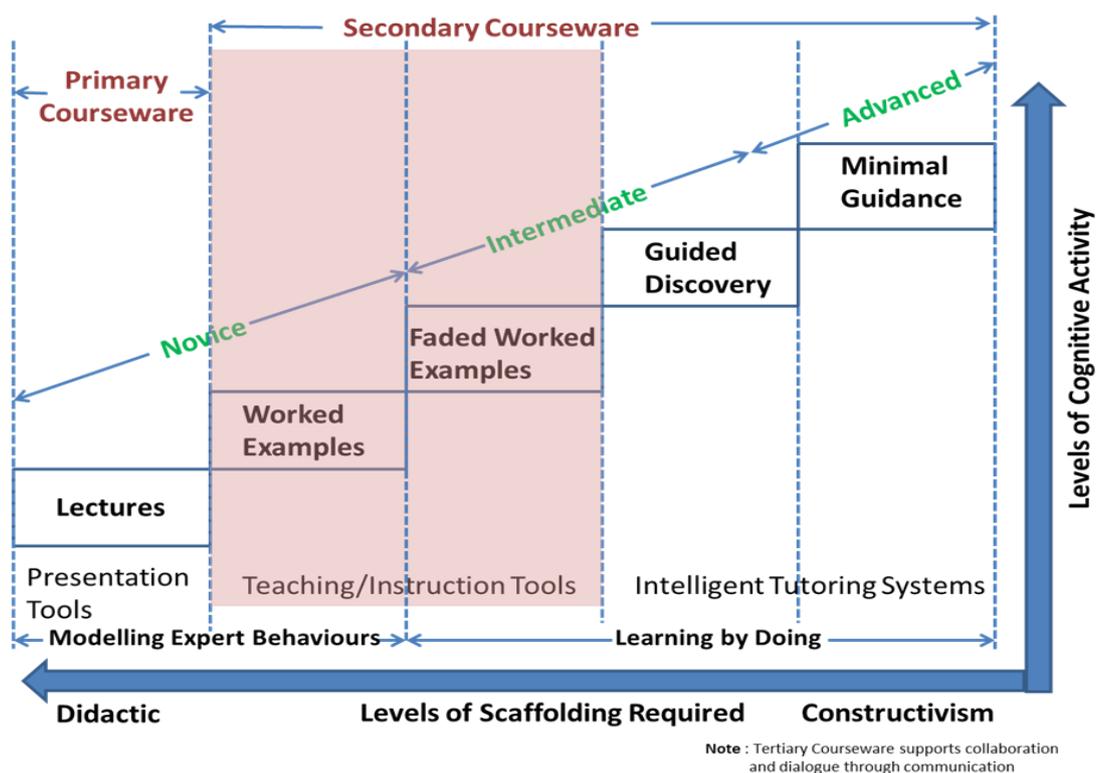


Figure 1.1 Relationships between Cognitive Activities, Pedagogical Methods, Different Levels of Learner and Courseware

Figure 1.1 shows the different pedagogical methods used to scaffold students' learning processes, recognising that students go through learning phases, from novice, to intermediate, to advanced. Supporting different levels of students' learning should depend on the varying levels of cognitive activity. The diagram also distinguishes how to use technology to support the different levels of learning, due to the one-to-many relationship between teacher and students.

Prior to the worked examples phase, shaded in Figure 1, key concepts in the problem-solving domain must be introduced. In higher education, this is often achieved using lectures and presentation software such as PowerPoint, referred to as primary courseware.

Building on the core concepts, the worked examples phases are used to initially develop and then consolidate students' knowledge of recurring problem-solving procedures. Fully worked examples, where every step is fully explained to the student, are the starting point, to be followed by *faded* worked examples, where some steps are partially explained, or not explained at all, to challenge the student and further embed their learning. Both fully and faded worked examples follow a set of solution steps that have been developed in advance by the teacher, modelling the expert's thinking process. Multiple worked examples should be studied so that the novice/intermediate can develop an appropriately broad experience of problems in the domain.

Once a reasonable foundation of problem-solving experience has been developed via worked examples, the intermediate learner should move on to a more un-scaffolded stage where the emphasis is on attempting to solve problems, rather than studying problem solutions. This is typically carried out in two phases, the first known as *guided discovery*, where the student is in control of directing which solution step to pick next. The student is then given feedback on the choice, either by a human tutor or by a so-called *intelligent tutoring system*. It is up to the student to decide on the next course of action based on this feedback: they may retrace their steps, realising they have made a mistake; or else they may continue, both in the case where they are moving correctly, or when they are not aware of a mistake they have made. The support system, whether human or technology-based, will continue to give them the best feedback and advice it can, in whichever case.

Finally, once the intermediate has further developed their bank of solved problems, as well as practised, with guidance, the problem-solving steps they had seen in the worked examples phase, they have reached an advanced stage and are ready to work on problems with no support at all - the *minimal guidance* phase. They now have sufficient experience to tackle most problems they face,

with each new problem solved being accommodated into their existing well-structured bank of domain problem solving knowledge.

The research in this dissertation does not concern the development of intelligent tutoring systems. Instead, the focus is on the development of technology to assist in the construction of interactive worked examples, both faded and full.

To summarize, these five learning stages very clearly show that different levels of learners should be taught by using different pedagogical methods. There is no single method which could apply to all the stages of learning. As the learner's level increases, the teacher's guidance should be faded gradually and learner's cognitive activity should be increased gradually. Hence, to support learners to achieve their learning target by using computing technology, the worked examples delivered should be in varied forms that fit their learning level. The role of computing technology in different levels of learning is also different.

In this research project, the major focus is on promoting and leading novice learners through the second and third stages to become intermediate learners. The role of computing technology should focus on supporting the creation of full and faded interactive worked examples. Therefore, an authoring environment for teachers to produce suitable interactive worked examples to support these two learning stages has been created. This dissertation does *not* concern the development of intelligent tutoring systems.

1.3 The Thesis Statement and Research Questions

The thesis statement is:

A usable authoring environment for delivering interactive worked examples can be developed that:

- a. delivers best practice interactive worked examples to students in a computing science context;*
- b. enables teachers to create such interactive worked examples without having to engage in bespoke programming;*
- c. facilitates evolution of them on the basis of feedback from the students.*

Based on this thesis statement, three research questions were identified:

1. Can an authoring environment be built to support the production of best practice interactive worked examples?
2. Can the authoring environment be used to deliver interactive worked examples?
 - a. For teachers, is the system sufficiently usable that they can create examples following the best practice recommendations based on education research results?
 - b. For students, are these delivered examples usable?
3. Can teachers create new examples or evolve existing examples over time to match the dynamic nature of teaching, on the basis of students' feedback and usage data?

Research Question 2a is different from Research Question 1. We could answer Research Question 1 positively but not have a system that teachers can use effectively.

The process of verifying this thesis statement comprises six activities. These are outlined below, together with an indication of how they contribute to answering the research questions.

1. Identify best practices for designing worked examples from the literature review, which is discussed in section 2.5.3 of chapter 2. This contributes to Research Question 1 (RQ 1) - identifying requirements for IWE and RQ 2a - use of best practices.
2. Review the literature about usability in order to define a suitable strategy for assessing the usability of the new authoring environment (IWE). This is a prerequisite for RQ 2b.
3. Develop a prototype of IWE and evaluate the usability of interactive worked examples during iterative development. Feedback about these examples can be collected from students and real life requirements can also be identified, which could guide further development of IWE. This contributes to RQ 1.
4. Ask 2 or 3 teachers to create interactive worked examples using IWE in order to evaluate its usability and identify real life requirements, which could guide further development of IWE. Then, ask 1 or 2 teachers to use it as part of their on-going teaching. These evaluations aim to

demonstrate that IWE can be used by teachers across different topics in computer science. This contributes to RQ 2a.

5. Deliver these interactive worked examples to students and evaluate their usability through analysing data by using the methods discussed in chapter 3. Contributes to RQ 2b.
6. Evaluate the feedback mechanisms with the aim of demonstrating that interactive worked examples can be developed and evolved over time by individual teachers based on students' responses and their usage data. Contributes to RQ 3.

1.4 Contribution of the Research

IWE demonstrates that it is possible to create or modify interactive worked examples without bespoke programming and that the worked examples produced are usable by students. It reduces the difficulty and cost of creating and modifying interactive worked examples.

A data model to capture the textual and/or graphical detail of worked examples suitable for the kind of problems identified in this thesis is defined. A dynamic process model for how these data can be presented to students is also defined, so that they can explore the worked examples in a step-by-step fashion, with specified parts of the worked example being shown and hidden on each step as appropriate.

The capacity for the tool to capture data on how students are working through the worked examples enables a level of so-called *learning analytics* [10]. A teacher can analyse the usage data to determine how often a worked example is used, whether all of it is viewed or only a part and how long is spent at each step, which parts may be causing difficulties, and therefore what action to take. For example, if the majority of students pause for an unexpectedly long time on a particular step, or get one of the questions wrong, the teacher can adjust the worked example, discuss the issue in class, and/or change their teaching approach for the next run of the class, as appropriate. Compared this with a standard text-book or on-line PowerPoint presentation, the former, no usage data whatsoever can be collected, beyond perhaps borrowing figures in a library;

the latter, the best a teacher can expect is a count of the number of downloads of the resource.

The intended teacher users of IWE will benefit by being able to create and modify interactive worked examples themselves without having to purchase ‘off the shelf’ ones or pay a third party to develop and modify examples. The disadvantage of using ‘off the shelf’ examples, even if they are free, is that teachers have to adapt their teaching to fit around these inflexible examples.

1.5 The Structure of Thesis

Chapter 2 expands on the context of the research by presenting an expert’s approach to solving a typical transformation-based problem in Computing Science, which might be presented as an interactive worked example. It then reviews research results related to problem solving, focussing on Human Cognitive Architecture, Cognitive Load Theory and the Cognitive Apprenticeship model. A critique of current instructional designs is given, followed by a review of research into the value of worked examples in education. The value of interactive worked examples is discussed and the disadvantages of the current technology for producing them are highlighted. This leads to the conclusion that an authoring environment for the production of interactive worked examples would be valuable and identifies the overall requirements for such an authoring environment (IWE).

Chapter 3 reviews the literature of usability evaluation as the other part of the background. It focuses on how to use these evaluation techniques properly in order to provide theoretical guidance for evaluating IWE. As IWE is designed for education perspective, only applying traditional techniques from human-computer interaction point of view is not enough; cognitive load theory should also be considered. Finally, the strategy for evaluating IWE is described.

Chapter 4 describes the detailed requirements for the authoring environment. It includes the architecture of this environment, the basic concepts used to build it, and the data model required together with the detailed use case descriptions for the two types of users, teachers and students.

Chapter 5 describes the design and implementation of this environment. The design part includes the initial user interface designs for the teacher and the student to access the system. The implementation part illustrates the final implementation results of the user interfaces, with comments on changes that were made to the initial designs.

Chapter 6 describes the evaluation results following the sequence of evaluations and surveys carried out during the iterative development process, from the initial feasibility study to identify the usefulness of a prototype environment, to the investigation of how target users actually used the system. It also identifies the short-term targets to be improved for carrying out a field study for further evaluating the usability of IWE. Longer-term targets for improving IWE are also reported.

Chapter 7 describes a field study which was conducted in an introductory programming course for first year university students. It demonstrates the usefulness of IWE from two aspects, which are usability and utility of IWE. It addresses the general usability of IWE by students, using the SUS questionnaire, and then investigates how useful IWE is in a real education context, for example, how the teacher can use feedback to evolve the worked examples that were delivered.

Chapter 8 addresses the research questions presented above and discusses how well IWE and its evaluations provide support for the thesis statement. It also identifies possible future directions for widening the scope of this research.

Chapter 2 Education Background

2.1 Educational Context for the Research

Teaching and learning computing science knowledge and skill is recognised to be challenging. For students, it requires correct understanding of abstract concepts. For teachers, it not only requires effective explanation of abstract concepts, but also demonstration of how to apply these abstract concepts to solve problems [11].

2.1.1 Transformation-based Problems

Problems to be solved in computing science often involve the analysis and transformations of representations, for example, text and diagrams, into alternative formats. For example, a database system is built from specific requirements described in human language. The requirements are analysed by a human first, and then transformed, for example, into a graphical design language, such as ER diagrams. Later on, these diagrams will be transformed into SQL language, to become readable by machines. In the programming context, in order to solve a problem described in human language, a plan is required based on analysing the problem text logically, and then a programming language is used to implement this plan. In a software engineering context, requirements are transformed into a Use Case diagram and definitions first, and then the Class diagram can be designed based on the use case analysis. Later on, the designed Class diagram will be implemented using a programming language, which is another transformation. These are all typical problems in computing science, involving the transformation of the real world representations of the problem into representations in formalised language, or between representations in formalised language. The arrow in figure 2.1 depicts this fundamental aspect of all computation.

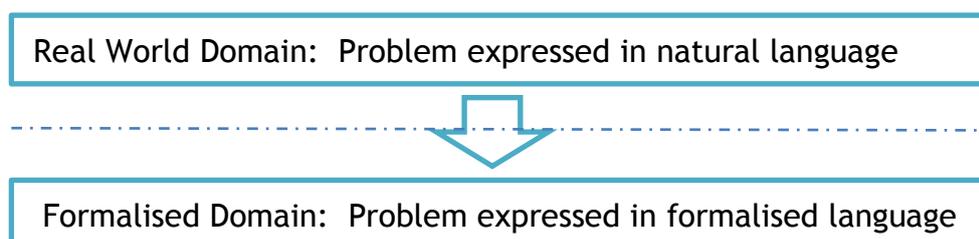


Figure 2.1. Transformation Problems in Computing Science

The processes involved in developing a model, including generating abstractions and making decisions, do not follow a formulaic thinking process. Instead, a number of guidelines are typically produced to aid the transformation process, but previous experience also plays a crucial role during the process.

To explore this kind of problem solving in more detail, a typical computing science example is used to explain and demonstrate in detail an expert's thinking process while solving a problem. By doing this, it should be clearer why some of the steps in the transformation process require judgement and hence cannot be completed algorithmically.

2.1.2 Solving Transformation-based Problems Requires Judgement

Building an ER diagram is an essential skill for students in a database course to develop. Consider the following example that might be given to a student: “A company has a set of departments. Each department has a name, number, manager and possibly several locations. The manager started managing the department on a given date. A department controls several projects, each with a name, number and location. Each employee has a name, address, salary, supervisor, department, gender, date of birth and national insurance number. An employee may work on many projects, not all in their own department, and works a numbers of hours on each of these projects. Please draw an ER diagram to represent the structure of the company. State any assumptions that you make.”

To explore the expert's thinking process in building an ER diagram for this scenario, an expert with experience in building commercial database models and experience of teaching database courses was asked to work through this example and explain his thinking. He wrote notes on a whiteboard while talking through the steps he was taking. The researcher recorded the expert's reasoning as he solved the problem, but also asked questions to clarify points that were made, if necessary. In the following record of the process, the expert's comments are shown in italics, with the researcher's commentary on the expert's comments added afterwards.

A common guideline suggested for starting the process of creating an ER diagram from a problem description is to identify the nouns, which are possible entities or attributes of entities, and the verbs which are possible relationships. However, these are only guidelines not rules. Listing every noun and verb found in the description is laborious and is often more confusing than helpful.

It is clear that the expert is aware of the difference between rules and guidelines, and that the rule, commonly used in education to get novice database designers started, of writing down all the nouns as possible candidates for entities, is not that helpful. In particular, the list can become long and unwieldy.

The recommended approach to this type of problem is to read the whole scenario right through and get an overview of exactly what is required. In this example, it is the penultimate sentence that gives us this: we must build a database model for the company described earlier in the problem text. The final sentence suggests that the scenario is not precise and there may be the need to identify ambiguities and makes assumptions to complete the exercise. This will probably only be picked up by the best students.

This is more than expert teacher speaking, recognising that novices often are not clear about exactly what it is they've been asked to do. The recommended approach can be found in, for example, [12] pp 1326-1327.

Start by looking for the possible major entities in the description.

An entity is defined as a group of objects with the same properties, which are identified by the enterprise as having an independent existence, which can be objects with a physical (or 'real') existence or objects with a conceptual (or 'abstract') existence [12] (p 343). Determination of what is a major entity is made partly on the basis of the definition given above and partly by experience, recognising, or having a feel for the kinds of items that should be modelled as entities.

Starting from the beginning of the problem description and looking for physical (real) objects in the scenario that could be entities. The first possible entity is 'company', but this is not an entity because it is the context for the data model, the ER diagram should represent the structure of this company.

In discussion, the expert noted that another way of determining whether this is an entity is to consider how many instances of this entity there would be in the final database - and if there is only one, then it is typically not an entity. For company there is clearly only one.

The next potential entity is 'department', which relates to a real thing and because there are several of them. The next sentence suggests a number of possible candidate attributes: name, number, manager and one or more locations. Locations will be a multi-valued attribute.

Items such as name and number help describe, or are a part of, the larger thing - the department - and don't have an existence in isolation. How does the expert know that *name* does not have a separate existence? Contributing evidence is: a department typically has just one name; one might write "the name of the department is", showing that the name *belongs* to the department; and the expert will have seen countless examples like this before.

Assume that each instance of a department can be uniquely identified by its number (the primary key), another essential property of an entity. For the time being, ignore the next sentence, it doesn't identify another possible entity.

In fact, this next sentence is describing a relationship between managers and departments but manager is unlikely to be an entity because normally managers are employees and likely to be a subset of a larger entity.

Moving on, the next interesting thing is 'project', which is a possible entity with attributes: name, number and location and assume that project number is the primary key.

This one seems straightforward.

The next sentence identifies 'employee' as another potential entity with possible attributes: name, address, salary, supervisor, department, gender, date of birth and national insurance number. Name, address and date of birth might be composite attributes, consisting of a number of components. It isn't possible to guarantee that employee's names are unique but the national insurance number will be a unique identifier for each employee.

The thought-process of the expert is clearly shown here, in the way that they are imagining the need to uniquely identify the instances of an entity.

The final sentence describing the company structure (“an employee may work ...”) does not identify any new entities but will be important for identifying relationships.

So the current thinking can be summarised as that there are three major entities with their possible attributes listed in the brackets:

***Department** (Name, Number - primary key, Manager, Location -"possibly several" - multi-valued)*

***Project** (Name, Number - primary key, Location)*

***Employee** (Name - probably composite, Address - probably composite, Salary, Supervisor, Department, Gender, Date of birth - probably composite, NI number - primary key)*

Note how the descriptions of the three major entities (as written on a whiteboard as the expert worked through the problem) contain further information in the form of notes: primary keys are noted; "possibly several" indicates that this could be a multi-valued attribute. So, the expert is thinking at many levels simultaneously, recording detail that will be required later.

The guidance that verbs represent relationships between entities is useful but not all verbs define relationships and not all relationships are identified by verbs! So, work through the problem description again from the beginning.

This again identifies the need for experience, and that there is no easy rule such as "verbs identify relationships" that can be followed by a novice.

The fact that the company is organised into departments gives no information about relationships. In the next sentence, the verb ‘has’ identifies the attributes of Department. Again, the next sentence will be skipped for the time being as it does not explicitly link two of the candidate entities identified. The next interesting verb is ‘controls’; this identifies a straightforward relationship between one Department and a number of Projects, so controls is a 1-N relationship.

The expert notes that certain words are *interesting* - and this will be on the basis of experience having seen problems of this kind. He knows that *controls* is exactly the kind of word that defines a relationship between two entities. Furthermore, he knows to ignore some sentences as irrelevant to describing relationships - here, the first two sentences. The third sentence is skipped for now - the expert knows it is important, but is choosing to focus on finding relationships that directly link the main entities. This is a clear guideline, though probably often unstated.

'Each employee has a name, ...' simply gives information about the possible attributes of an entity. In the next sentence, 'work on' identifies another relationship, between the entities Employee and Project, define works_on as a M-N relationship because an employee may work on a number of different projects and a project may involve several employees. The hours an employee works on a particular project will be an attribute of the relationship. The statement 'not all in their own department' means that the relationship works_on is independent of any relationship between Employee and Department.

Identifying this relationship is quite straightforward, but there is subtlety to it. Picking up on the clause "not all in their own department" is essential to properly modelling the company.

However, there are still some further questions to resolve. There should be a relationship between the entities Employee and Department.

The expert is recognising, based on experience of this kind of problem, that the ER model needs an additional relationship, although it does not jump out of the problem text. It will not always be the case that this third relationship will be present, but the expert's intuition, based on experience, is that it should be there.

In the current state of the ER model the relationship between Department and Employee is represented by department (number or name) being an attribute of Employee and the statement 'not all in their own department' indicates that each Employee belongs to a single Department. A 1-N relationship Has_staff should be defined between Department and Employee, and the attribute department deleted from Employee. Storing the department as an attribute of Employee could cause problems because there could be values for the department attribute that do not relate to any Department entity.

Note here that the expert is able to discern the necessary relationship from the possible attributes so far defined, made easier by the fact that the attribute name is the name of one of the main entities.

Now return to the third sentence where the verb 'managing' identifies a relationship between manager and Department but currently manager is an attribute of Department. Assume that every manager of a Department is an employee of the company and each manager manages only one Department. A relationship manages between an Employee and a Department can represent the role of the manager; it will be a 1-1 relationship and the date the manager started managing the Department

will be an attribute of the relationship. Delete manager from attributes of Department.

The expert is drawing on experience either of companies in general, or having seen problems similar to this one before. He knows that a manager will typically be an employee, and therefore that managing is a relationship between particular employees and particular departments.

One remaining problem is the attribute supervisor of Employee. Assuming that all supervisors are also employees of the company this will generate a recursive relationship between Employee entities. An employee may be the supervisor of a number of other employees (supervisees), so a 1-N relationship supervises can be created from Employee to Employee. Supervisor can be deleted from the attributes of Employee.

Again, the expert makes a sensible assumption about supervisors.

At this point, three entities and five relationships have been identified:

***Department** (Name, Number - primary key, Location - "possibly several" - multi-valued)*

***Project** (Name, Number - primary key, Location)*

***Employee** (Name - probably composite, Address - probably composite, Salary, Gender, Date of birth - probably composite, NI number - primary key)*

Department Controls Project: 1-N

Employee Works_on Project: M-N with attribute hours_worked

Department Has_staff Employee: 1-N

Employee Manages Department: 1-1 with attribute start_date

Employee Supervises Employee: 1-N

So a draft ER diagram to represent the analysis above can be produced, [which is shown in Figure 2.2]. Note that role-names have been added to the relationship Supervises to clarify that 1 supervisor Supervises N supervisees. Attributes of entities have not been added to the diagram as they obscure the structure of the diagram. However, the attributes of the relationships Manages and Works_On have been shown as these will affect the transformation of the ER Diagram into tables.

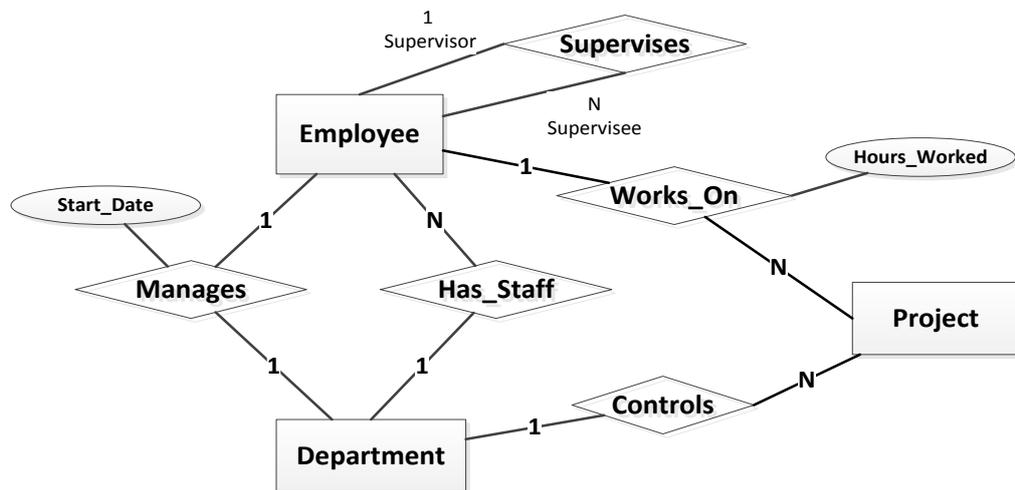


Figure 2.2. A Draft Design ER Diagram

This ER diagram [given in Figure 2.2], together with the definitions of the three main entities and their attributes, would be a reasonable answer from a novice student. However, to fully define the database model, there are still issues to resolve, for example, optional versus mandatory relationships. Points that might be raised here are:

- Does every Department control at least one Project? There may be administrative departments that do not control projects but we assume that every project is controlled by a Department. So, **controls** is probably optional - mandatory.
- Does every Employee work on at least one Project? Again, there may be administrative staff members in a Department who are not involved in Projects. However, each Project is assumed to have at least one Employee working on it. So, **works_on** is optional - mandatory.
- Assume that every Department has Employees and all Employees belong to a Department. So, **has_staff** is mandatory - mandatory.
- Not all Employees will be managers but every Department has a manager. So, **manages** is optional - mandatory.
- It is not possible for every Employee to be a supervisor nor can every Employee be supervised by another Employee. So, **supervises** must be optional - optional.

Based on these assumptions, a second draft design with participation constraints can be produced, as follows [shown in Figure 2.3]. This would be an excellent answer from students.

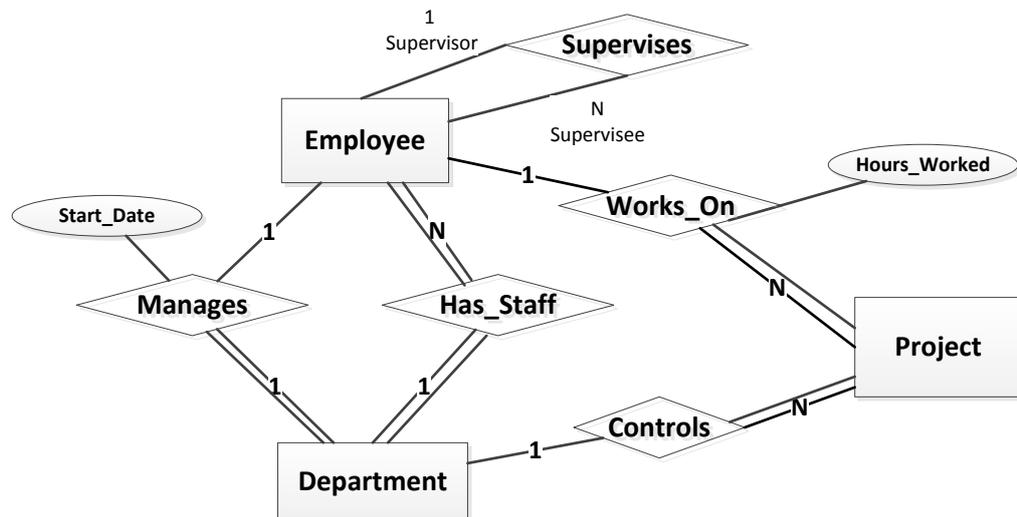


Figure 2.3 A Second Draft Design ER Diagram with Participation Constraints

There is one other possible complication with this model (based on my experience but students would not be expected to identify this) concerning the attribute location of Department and Project. If the company is a service company with several service departments that provide technical support at client locations then there will be no overlap between the values of these attributes. Alternatively, all projects might take place at sites where Departments are located, in which case there is only one set of location values for the two attributes. Other decisions about the exact formats of possible composite entities would need to be considered when database queries are specified.

The expert also added that the next stage in the process, transforming the ER diagram into a set of SQL tables is more straightforward because it involves transforming one formalised description into another. An expert will know patterns for transformations of: composite attributes, multi-valued attributes and relationships (1-1, 1-N and M-N) possibly with attributes, into tables.

2.1.3 Judgements Require Prior Successful Experiences

This example highlights many of the issues in the transformation process of solving problems in computing science. The demonstration of an expert's thinking process shows that the whole process of solving problems requires judgments and decision-making. Problem descriptions are written in natural language and imprecise. So, an algorithmic approach to creating a solution is not possible and a heuristic approach has to be used. Complex decisions are taken on the basis of previous experience: an expert will have seen many problems before. So, after reading the problem description, he can match the current

problem to previous related problems, which have already been solved, and reuse whatever solution methods were used there.

Examples of such decision-making are:

- The company is not an entity, as it is the context of this problem.
- Manager is not an entity as it is assumed all managers will be employees. Therefore manager is not an attribute of the department, and “manages” will be a relationship between department and employee. Creating an entity called “Manager” would not be wrong but in this case it is unnecessary. It would create two one-to-one relationships between Manager - Employee and Manager - Department. When implemented Manager will contain the same values as the relationship manages.
- The decision on how to model the location of Department and Project depends on how locations for departments and locations for projects are related to one another, which is not specified in the scenario given.

Note in particular that it would be hard to codify the knowledge needed to make these decisions into a set of rules. Take the first example, concerning the observation that some items that look as though they might be entities, such as company, but are not. The rule might be: “Discard a candidate entity that appears to have only one instance and captures the entire system being modelled.” This is much more easily understood with the aid of several examples, until the rule alluded to above is formed. The issue with manager could be formulated as “if an attribute of an entity is also an instance of another entity then this should be represented as a relationship”. The issue with locations could be formulated as “If any attribute name appears twice, determine whether they are the same or different attributes, depending on the context.” Again, these are much more easily understood with several examples. Finally, a rule widely used in database system teaching is to “make a list of all nouns and consider these as possible entities or attributes.” The example shows that this is not a good rule, since the context in which a noun is used is very important. For example, in the above scenario there are three occurrences of the noun ‘name’, but from the context it is obvious that they are different: the Department name, the Project name and the Employee name, which are independent sets of values. The other widely-used rule is that “verbs can be used identify relationships

between entities”. In the example, only two of the verbs identified relationships and three other relationships were identified by analysing the implicit relationship between attributes and entities.

Therefore, the challenge of the transformation from a representation to another representation is that it is non-formulaic. There is some guidance that can be followed, but proper use of the guidance to generate abstractions and make decisions is highly affected by previous experience. Experts have seen enough prior examples of relevant problems that they have developed generic patterns for each related category of problems.

A key aspect of educating students therefore is to help students to build up successful problem-solving experience. But how are the generic patterns necessary for problem-solving developed and stored by students? To explain this, it is necessary to understand the process of learning at a cognitive level, and this is introduced next.

2.2 Understanding Learning at the Cognitive Level

In this section, the human cognitive architecture is described in order to understand the processes occurring in the human memory system that contribute to learning. Cognitive Load Theory (CLT) builds on the human cognitive architecture model to demonstrate why some learning designs are more effective than others, with reference to the limitation of the human cognitive architecture.

2.2.1 Human Cognitive Architecture and Schema Development

The process of learning involves gaining new knowledge and skills, and assimilating them alongside existing knowledge and skills. This means that the brain needs both to store items and then reuse them in the right situation. Hence, it is worth knowing at the outset how the brain processes information in order to store and reuse stored information. Atkinson and Shiffrin [13] proposed a psychological model which describes the structure of the human memory. In the Atkinson-Shiffrin model, the human memory is divided into three sub-

components, which are sensory memory, working memory and long-term memory.

- Sensory memory is the shortest-term element of memory and retains the impressions of sensory information, which come from the human senses. It is considered to be outside of cognitive control and is instead an automatic response, and is not considered here.
- Working memory is generally considered to have limited capacity. It has the ability to remember information over a brief period of time, but it is long enough to be used for further information processing, such as internally repeating or refreshing the information in working memory, or then passing it on to the long-term memory, described below.
- Long-term memory is the element of memory in which associations among items are stored. It provides the lasting retention of information, from minutes to a lifetime and seems to have an almost limitless capacity.

Sweller introduced the concept of a schema as a cognitive structure for storing information in long-term memory [14]. A schema is a particular structuring of information and experts in a discipline have structuring that enables highly effective retrieval of the stored information. They associate elements together so that the collection can be remembered as a single element. For example, the number 100 is recognized as a single element, not as three separate digits. The same principle underlies the mind-mapping revision technique [15] of condensing written notes repeatedly, using a tree structure of headings to remember a large body of related elements.

Sweller [16] also gave his definition of working memory, long-term memory and schemata from the angle of learning and proposed a natural information processing system. Working memory is a structure that processes information coming from either the environment or long-term memory and that transfers learned information for storage in schemata into the long-term memory. Long-term memory has a massive capacity for information storage. It contains cognitive schemata that are used to store and organize knowledge by incorporating multiple elements of information into a single element with a specific function.

Cowan proposed an integrated model of working memory in which some of the representations (schemata) held in working memory are an activated subset of the representations held in long-term memory. In Cowan's model, working memory was organized in two embedded levels. The first level, called activated memory, consists of an unlimited set of long-term memory representations that are activated. The second level, called the focus of attention, holds up to four items or chunks of information from the environment [17]. This second level is the central limitation of the working memory system.

In the problem solving domain, a schema represents a category of problems as a set of key characteristics of such problems and captures the variations that have been seen before. Each new problem belonging to this category can be stored under the same schema. Furthermore, the method for solving problems of this category is also stored with the schema[18].

With such schemata, problem solving is a process of recognizing the essential characteristics of the new problem, and then matching these against the characteristics held within each problem category schema. Cowan's model indicates that an expert can hold the activated schemata of a very large number of previously seen problems in working memory alongside the relatively few key characteristics of the new problem, and therefore perform the necessary pattern matching, as all necessary data is in the working memory.

Studies of chess grandmasters [19] and also Soloway's work on programming comprehension [20] back up this view by showing that experts solve problems better than novices when those problems follow well-known patterns, but not when they are random problems adhering to no commonly-seen pattern. It is this pattern-recognition based on acquired schema that makes for expert behaviours.

If the learning result is to alter the long-term memory, it is important to know how the learning process is achieved. Kirschner and Van Merriënboer [21] described learning in terms of four processes involving the creation and modification of schemata. This can be combined with Adaptive Control of Thought-Rational (ACT-R) framework [22] to give five stages in schemata development.

1. Prior to schema construction: many separate examples are seen and recorded in long-term memory independently. A new example is related to these stored examples in an attempt to recognize by analogy. Learners' performance is slow, error prone and working memory load is high, because the details of the new example must be compared with the details of every stored examples. For instance, the experiences of seeing many different animals are all stored separately, and when a horse is seen it is compared against all examples of animals.
2. Schema construction: repeatedly seeing related examples and recognising features common to all causes a new schema to be formed. For example, the numerous instances of different horses (e.g. with different sizes and colours and names) are stored under a schema for 'horse' with the common characteristics of e.g. four long thin legs, long straight-haired tail, long neck and nose, long-haired mane, can be ridden, with hooves, and so on.
3. Schema assimilation: new elements of information are incorporated into existing schemata. For example, a child seeing a zebra for the first time and calling it a horse, associating it with the horse schema.
4. Schema elaboration: elements consisting of lower level schemata are combined into higher level schemata building increasing numbers of ever more complex schemata. The child takes into consideration the different properties of a zebra compared to a horse, perhaps calling a zebra a horse with stripes. In this case, the horse schema would have a sub-schema of horse with stripes. Of course, the child may choose other classifications, for example, wild animal, separating out common and different features of a zebra compared to the more generic wild animal.
5. Schema accommodation: existing schemata based upon recurring new information which are incongruous or inconsistent with existing schemata are adapted. When the child eventually learns the name of zebra, this information is accommodated, perhaps adjusting learning information about horses to now have a "horse-like" schema, with sub-schemata for horses and zebras.

Because a schema can be treated by working memory as a single element if a schema has become sufficiently automated after long and consistent practice, the limitations of working memory disappear for more knowledgeable learners

when dealing with previously learned information stored in the long-term memory.

2.2.2 Cognitive Load, Cognitive Overload, and Learning

Cognitive Load Theory (CLT) [14] builds on the notion of a limited working memory capacity and a vast long-term memory capacity as the model used for understanding the human cognitive architecture. It is a set of learning principles that deals with the optimal usage of the limited working memory. Sweller [14] and Cowan's work [17] proposes that since working memory for new information is limited, if the complexity of instructional materials is not properly managed, this could result in cognitive overload for learners. Tuovinen and Sweller [23] and Pass et al. [24] further develop this theory and suggest the free exploration of a highly complex environment may generate a heavy working memory load which is harmful for learning. Especially it is not good for novice learners, due to their lack of proper schemata, which are needed for the integration of the new information with their prior knowledge. Novices, not possessing appropriate schemata, are not able to recognize and memorize particular problem configurations. Sweller et al. [25] also found that cognitive overload can impair schema acquisition, later resulting in a lower performance.

CLT [16] distinguished three types of cognitive load that occur in working memory during learning, which are:

- **Intrinsic Cognitive Load** refers to the number of elements that must be present simultaneously in working memory for a concept that is to be learned to be understood. For a particular learning task, the relative numbers of new elements and elements drawn from long-term memory are dependent on the learner's degree of prior experience (what is complex for a beginner is simple for an expert).
- **Germane Cognitive Load** results from active schema construction processes and is the result of beneficial cognitive processes for learning. It is effective cognitive load. For example, explaining the material to oneself, or rehearsal from practice repetitions.
- **Extraneous Cognitive Load** is the result of instructional techniques that require learners to engage in working memory activities that are not

directly related to schema construction. It can be caused by an inappropriate presentation of the learning material or by requiring students to perform activities that are irrelevant to learning. It is ineffective cognitive load, from the point of view of learning. For example, visual search processes for information during learning.

Based on CLT, learning outcomes are optimized when cognitive load fully utilizes the capacity of working memory with elements that allow for optimal schema acquisition. Too little or too much cognitive load results in low learning outcome. Optimising learning is a question of balancing, not minimizing or maximizing cognitive load. However, a reduction of extraneous cognitive load frees working memory capacity to be used for germane learning processes. If intrinsic load is low, learning can be successful despite a high extraneous load, although the exercise may seem tedious or boring. The total amount of cognitive load required for a learning activity needs to remain within the working memory capacity.

Based on this review of human cognitive architecture and cognitive load theory, supporting novice students to develop schemata in an efficient way should be the target for instructional design. Problems cannot be solved without proper and sufficient schemata; students must see enough problems and solutions and build enough experience in order to develop generic schemata. However, how can students develop these generic schemata efficiently, and crucially, how can cognitive overload and consequent wasted effort be avoided? This leads to an exploration of apprenticeship models.

2.3 Cognitive Apprenticeship to Develop Schemata

Traditional apprenticeship learning is based on learners gaining significant experience of problem solving before being asked to solve problems themselves. The master guides the apprentice through a series of experiences that enable them to develop the necessary schemata to become a master[26]. However, apprenticeship, as a form for producing knowledgeable and skilled persons, has been overlooked in the modern education system, although it is still a key learning mechanism[27].

Collins et al. [28] introduced cognitive apprenticeship that utilizes the underlying principles of traditional apprenticeship learning. The traditional apprenticeship assumes that apprentices can learn domain-specific skills and processes through a combination of observation, coaching and practice. Cognitive apprenticeship emphasizes that the focus is on cognitive skills and processes, and it is designed to make the master's otherwise-hidden internal cognitive processes externally visible for students to observe and to imitate.

The theory of cognitive apprenticeship holds that masters of a skill often fail to take into account the implicit processes involved in carrying out complex skills when teaching novices. Cognitive apprenticeship is designed to bring these tacit processes into the open, where students can observe, enact, and practice under the teacher's help. Three core instructional methods in cognitive apprenticeship are modelling, coaching, and scaffolding which are designed to assist students in acquiring an integrated set of cognitive skills through processes of observation and of guided and supported practice. Modelling occurs when experts describe their own cognitive processing in the course of carrying out a task. Coaching occurs when an expert offers hints, comments and critiques to a student who is carrying out a task. Scaffolding is the support given during the learning process, which is tailored to the needs of the student with the intention of achieving the learning goals. Scaffolding is needed when a student is working on a task but is not yet able to successfully manage each part without some kind of support. The whole point of scaffolding is for the expert to give just enough support to novices to help them until they know how to do it by themselves and then gradually fade this scaffolding away[28].

Collins et al. [28] concluded that cognitive apprenticeship is a useful instructional paradigm when a teacher needs to teach a fairly complex task involving judgement to students, but not teach basic skills or concepts. By properly applying this model into instructional design, teachers need to: "identify the processes of the task and make them visible to students; situate abstract tasks in authentic contexts, so that students understand the relevance of the work; and the diversity of situations and articulate the common aspects so that students can transfer what they learn." [28]

The cognitive apprenticeship model is an ideal model to be used for instructional design in computing science. When using it, a teacher needs to visualize the whole thinking process of solving a problem for the students as well as explaining the reasons for following this process in detail. This is helpful for novices to obtain initial experience of problem-solving, supporting schemata acquisition and development.

However, a typical one-to-one apprenticeship model, whether led by the master or one of the master's skilled colleagues, is too expensive to use, as it requires teachers spending too much time to teach and offer support as students need it. The teacher must be able to diagnose when a student needs support and know when to remove some support. Within the typical mass educational environment, the very limited numbers of experts cannot offer enough support for the potentially huge number of students' requests. Therefore, mass education instructional designs attempt to offer similar support to apprenticeship models but by other means.

The current instructional designs in computing science are now explored to determine how they attempt to satisfy the needs for supporting students' developing experience, in light of schema acquisition, cognitive load theory, cognitive overload and cognitive apprenticeship.

2.4 Current Instructional Designs for Transformation-based Problems

After knowing how students can develop and store schemata as the basis for developing problem-solving skills, let us review how the current prevailing instructional designs can help or scaffold students to do this.

In summary, the main way of students learning in mass education (e.g. university) settings currently is through absorbing the material from the teacher and bringing their questions back after individual study either in tutorial sessions or directly in a one to one session. The teacher will also ask the students to perform tasks in order to develop the students' problem solving skills. Some of these are assessed in order to check whether the students have acquired the knowledge and skills correctly.

This design can be viewed in the context of schema acquisition and development as follows. The instructional steps used are typically following the procedure of introduction, demonstration and exercise. The first two steps are the early stage for instructional design, as students are novices and mainly focus on acquiring knowledge. The third step is the intermediate or advanced stage for instructional design, where students practice applying their knowledge, and setting problems to be solved should be the main target. The teacher needs to assess students' ability of applying knowledge of solving problems to see students' learning results. The details of each step are described below.

1. Introduce core concepts in the topic area in the lecture. These concepts can be used to solve problems later on. Lectures provide a very low level of interactive opportunity, but are cost-effective.
2. Schema building process for novices, to develop problem solving patterns. Demonstrations designed to introduce typical problems and their solutions are often offered in three ways, which are:
 - a. In the lecture, the teacher demonstrates and explains live worked examples. Students passively observe these problem solving activities with very limited interaction. During the explanation process they can ask questions directly, in practice very few do.
 - b. After the lecture, students can actively read worked examples from text books or teacher's hand-outs to build up their experience of problem solving.
 - c. In tutorials or labs, the tutor often focuses on explaining live worked examples, which are often small practice problems from the tutorial sheet. Students can obtain some immediate support while working on similar small practice problems.
3. Students do exercises requiring them to practice these problem solving skills, from small to larger problems. The process aims to foster schema elaboration and accommodation for intermediates. The technique of doing exercises can be:
 - Ask students to complete a few smaller problems initially, and then lead them to solve a larger problem.
 - Ask student to solve a new problem, based on their experience of solving previous or related problems.

- Give part of a solution, and ask students to complete the rest of the solution.
- Ask students to solve complete large problems from scratch, based on the experience gathered from the previous steps.

Having outlined the typical large-class instructional designs, the next section analyses aspects of the design in order to show that they are unlikely to support effective learning.

2.4.1 Weaknesses in the Current Instructional Design

2.4.1.1 Insufficient Worked Examples Leading to Cognitive Overload in Problem Solving

The current instructional design offers very limited opportunity for students to obtain experience of problem-solving directly from the teacher, the model in traditional apprenticeship. For example, they would ideally practise steps 2 and 3 in a one-to-one teaching scenario, as this would allow the student considerable opportunity for conversation and interaction to absorb experience of problem-solving from the teacher, but in practice this is infeasible.

Instead, demonstrations and worked examples are used to develop schemata for problem-solving. Sweller et al. [25] explained the *worked example effect* postulated with CLT. Because worked examples focus attention on problem states and associated operators (i.e. solution steps), enabling learners to induce generalized solutions or schemata for initial schema acquisition, it is more favourable to learn from examples with worked solutions than to solve problems from scratch.

Unfortunately, insufficient time is spent on working with worked examples, as evidenced in [18], [29], [30] and [31]. During the teaching process, how to make the judgements based on different situations and how to perform the transformations can be either tacit or poorly explained. Instead of concentrating on demonstrations of worked examples for novice learners, teachers direct students to attempt largely unguided problem solving tasks. Since the learners have not yet developed the necessary schemata, such problem solving is

cognitively extremely taxing on the learners, and consequently little if any learning can take place.

2.4.1.2 Worked Examples in Textbooks or Lectures are Incomplete

Even the few worked examples that are presented have problems. The worked examples in a textbook may not be as good as hoped. For example, Robins et al. [11] pointed out that typical introductory programming textbooks devote most of their content to presenting knowledge about a particular language. The prevailing textbook approach will help the students to understand the programming language and the structure of programs, but it does not show the student how to program. In another words, it does not reveal the problem-solving process. These static worked examples are useful for the presentation of a product (e.g. a finished program), but not for the presentation of the dynamic process used to create that product.

Typical worked examples, presented in a text book or a lecture, have several issues which are described below:

1. They do not allow for students to interact with or query the process. For example, students are reluctant to stop a lecturer to ask questions, because of limited time and exposure in front of the class; and a text book can seldom answer questions.
2. Text book versions of worked examples do not always necessarily fit into a particular teacher's teaching style pace. As the students' degree of prior knowledge is variable, the Intrinsic Cognitive Load of these worked examples does not suit every student. If unsuitable worked examples are provided, students will not be effectively engaged.
3. In a lecture or tutorial, some interaction can happen between the teacher and students. However, the teacher may not have enough time to clearly explain all steps of building the solution, or else this limits the complexity of the examples that are presented. A similar issue also exists for the worked examples in a text book due to the space limitation, which means the solution of the worked example is not presented in a very detailed style. Details of explanations for

every single step of the transformation are limited, and only a few snapshots of the transformation are provided, as it progresses.

4. Live sessions of explaining worked examples are not routinely captured for review by the students. The presentation vanishes as it takes place and nothing is saved afterwards. The explanations sound fine at the time; however, after the lecture it is hard for students to review live worked examples.
5. There are sometimes not enough suitable worked examples to provide proper support for all students, either in the lecture or the text book.

2.4.1.3 Wide Range of Learners' Prior Experiences are not Addressed

The worked examples selected for demonstration in the large mixed ability class should be carefully considered to match the majority of learners' prior experience. According to Laurillard [32], "students come to the classroom with a broad range of prior knowledge, skills, beliefs, and attitudes, which influence how they attend, interpret and organize in-coming information. How they process and integrate new information will, in turn, affect how they remember, think, apply, and create new knowledge." Kolb [33] suggested that it is not sufficient to have an experience in order to learn. It is necessary to reflect on the experience to make generalizations and formulate concepts, which can then be applied to new situations. They both pointed out that the learners' prior experience is an inevitable foundation for learning. Kalyuga [34] claimed that instructional designs should depend on learners' prior knowledge levels. Designs and techniques that are effective with low-knowledge learners can lose their effectiveness and even have negative consequences for more proficient learners. Hence, applying a single instructional design (e.g. just one worked example for a particular topic) is not able to satisfy all the students or accommodate prior knowledge. This means that the cognitive load is too high for the less experienced students and too low for experienced students.

2.4.1.4 Limited Dialogue between Teacher and Students

Laurillard [32] defines learning as the transformation of experience. She claimed that knowledge is information already transformed: selected, analysed, interpreted, integrated, articulated, tested and evaluated. She stated that the knowledge that students brought to a course would necessarily affect how they

deal with the new knowledge being taught. Teachers should often be aware of students' "likely misconceptions" about a given subject, based on available literature about teaching that subject, and prior experience. Teachers' efforts to provide learning guidance should include questions to elicit these errors and give opportunities for clarification. It is important to establish interactive dialogue between teacher and student to aid the learning process, which is necessary for "deep" learning to occur. The teacher should adapt explanations of the concept or actions in light of students' reflection or experience, which can allow students to produce the intended way of representing it. She emphasized the interactive dialogue between teachers and students should be continually accessible to each other to achieve the learning goal. Unfortunately, the current instructional design only offers very limited opportunity for teachers and students to have a continual interactive dialogue. This is not good for students needing to adapt their existing schemata or restructure schemata to create a new schema.

Limited dialogue between a teacher and their students reduces the chance to obtain feedback from each other. In general terms, feedback is any message generated in response to a learner's action. However, in this dissertation two types of feedback are defined, which is based on the students' interaction with the worked examples. These two different types of feedback also are also identified in Laurillard's conversation framework.

- Students' response is defined as follows: the students explore the worked examples with the explanations in order to achieve their learning targets; however, the worked examples are not good enough for their self-learning. Hence, they may have some questions after the exploration;
- Teachers' response is defined as teachers' action, which is based on the students' response. There are two possibilities, which are response to a group or class and response to individual. Group or class response is defined as if more students have similar questions, teachers could modify the worked examples to response to the group or class. Individual response is defined as if only very few students have questions, teachers could answer the individual question separately and directly. These

teachers' responses based on students' responses are aimed to help students to refine their concepts to finish the learning process.

It is challenging for a teacher to provide immediate feedback on individual responses, in, for example, a large or distributed class. The feedback system is bi-directional, and the following problems have been identified:[35, 36]

- Teachers cannot give enough explanations or feedback to all the students as they require it;
- Even though in some cases, the teacher can give enough explanations to all students as required, it is not usually a timely feedback in many cases;
- Teachers try to provide more explanations but, due to the lack of precise data related to students' learning experiences, it is very hard for teachers to provide the extra explanations.

It is also noted that the teacher has very little data of how a student experiences using worked examples or their performance of non-assessed exercises(i.e. which are not submitted for review). In a tutorial, a tutor can observe the performance of students, so some detailed problems can be identified and reported to the teacher. However, a successful tutorial depends on the experience of the tutor, the questions provided by the students and their willingness to participate in the tutorial process. Therefore, typically the teacher has a poor sense of their students' learning progress, because they have limited feedback on the students' usage of worked examples and non-assessed exercises. Furthermore, feedback from assessed exercises is usually too late to be useful for adapting current teaching. For example, feedback on student learning of material, introduced in lectures in week n , included in an assessed exercise due for submission in week $n+2$ and marked in week $n+3$, can only be provided 3-4 weeks after initial exposure.

2.4.1.5 Worked Examples are not Interleaved with Problem Solving

At the moment, worked examples are the major resource for students to obtain experience of problem-solving, and to generate schemata as patterns to solve problems in later exercises. Many students perform poorly while solving problems either on their own or in lab exercises [37].Among a number of

possible explanations for this poor performance is that current practice in using worked examples is ineffective, as will now be discussed.

According to [29] [38] and [39], it was found that using the usual procedure of instructional design, (introducing a topic, presenting one or a few worked examples, and then providing problems to be solved) is less effective than prolonging the phase of example study only and then moving to problem solving. Building on these findings, Trafton and Reiser [40] claimed the most effective way to use worked examples for acquiring new skills was to apply the knowledge gained to problem solving. Based on their two dependent measures: time to solution and accuracy of solution to solve problems, they suggested that worked examples must be available in memory during problem-solving. So they suggested that worked examples should be offered in a sequence, presenting a worked example which contains problem-solving guidance first, and then providing a similar problem to solve immediately afterwards, then another worked example, then a similar problem, and so on.

2.4.1.6 Summary of Issues with Typical Current Instructional Design

From the CLT point of view, in order to solve a problem successfully, students must have enough problem-solving schemata saved in the long-term memory. Unfortunately, the current instructional design does not expose many worked examples to the students, which limits learners' opportunity to develop initial schemata. Therefore, it does not work well in supporting students to obtain the essential experience of problem-solving. It jumps to getting students to solve complete problems from scratch, after demonstrating a very limited number of worked examples. Considering CLT, this jump is too large. So it is necessary to look for other supporting mechanisms to overcome these issues - how to help students to build experience efficiently is the major focus.

Worked examples are recognised to be an effective means of supporting cognitive apprenticeship, as they are built based on an expert's problem-solving experience [30], [31] and [41]. It is the way they are used in typical instructional designs that is at fault. Before considering how teachers can be supported to use worked examples more effectively, research on worked examples is

presented, in order to understand more deeply how they should be structured and applied.

2.5 Worked Example Research

A worked example is a step-by-step demonstration of how to perform a task or how to solve a problem. Typically it contains a problem formulation, solution steps and the final answer itself. For example, many worked examples can be found in mathematics or geometry textbooks, but they can also be used in other fields. Atkinson et al. [29] reported that worked examples could also be developed for music, chess, athletics, and computer programming learning. Worked examples are “instructional devices that provide an expert’s solution for a learner to study. Worked example research is a cognitive-experimental program that has relevance to classroom instruction and the broader educational research community.”

2.5.1 Worked Examples Effect

According to Kirschner et al. [30], worked examples constitute the epitome of strongly guided instruction and typically consist of a problem formulation, solution steps and the final answer itself. Worked examples make visible an expert’s problem-solving schemata to explain the steps of a solution for novices. Novices are provided with a worked example, as a schema of how to solve a problem, so they avoid having to engage in unnecessary trial and error processes [23, 24]. It also allows novices to circumvent most of the limitations of short-term memory, as their attention is focused and the relationship between problem solving steps is demonstrated.

Studying worked examples is an effective instructional strategy to teach complex problem-solving skills, it can be more effective than learning by problem solving [25]. Providing novices with worked examples rather than problem-solving exercises was shown to be a particularly powerful and efficient method for enhancing novices’ learning [25]. This can be explained by CLT. According to Kirschner et al. [30], solving a problem requires problem-solving search and search must occur in our limited working memory. Problem-solving search is an inefficient way of altering long-term memory, because its function is to find a

problem solution, not alter long-term memory. Problem-solving search can overburden limited working memory resources to be used for activities that are unrelated to learning (e.g. extraneous cognitive load). In contrast, studying worked examples both reduces working memory load because search is reduced or eliminated, and directs attention to learning the essential relations between problem-solving steps. Students learn to recognize which steps are required for particular problems, the basis for the acquisition of problem-solving schemata.

Reed and Bolstad [42] indicate that one example may be insufficient for helping students to induce a usable idea, and that the incorporation of a second example, especially one that is more complex than the first, increases students' learning outcome significantly. Therefore, the more worked examples students see; the more experience they have to build up their problem-solving skills.

Atkinson et al. [29] suggested providing multiple examples (at least two) of solving the same problem type improves learning and transfer before practising. Later on, examples and practice should be intermingled. This is further supported by Trafton and Reiser [40] who indicate that each worked example offered should be followed by a similar practice problem, since pairing each worked example with a practice problem produces better outcomes than a block series of worked examples followed by a blocked series of practice problems.

Sweller and Cooper [39, 43] demonstrated that learning from worked examples can be more effective than learning by problem solving, based on the experimental results that students learned more by studying algebra worked examples than by solving the equivalent problems. The finding can be explained by using the CLT. First, when a student, particularly one with low prior knowledge, reviewed an example, it helped to lessen cognitive load and maximize initial learning. Second, the cognitive schema created by the student while he or she was studying the example can then be used to deal with an isomorphic or similar problem to solve, for example, one with similar structure or elements to the example. The student can then easily recall the similar, just-reviewed example and does not need to grapple with many new and unfamiliar details in solving the new problem and searching through memory. At the same time, students can keep in mind active cognitive processing to strengthen the understanding of this type of problem in order to achieve deep learning.

Chi et al. [44] found that examples drawn from college level physics textbooks often do not include all of the reasons why a certain step in the solution was performed. So learners needed to work it out themselves and this phenomenon was termed the *self-explanation* effect. They discovered learners tried to establish a rationale for the solution steps by pausing to explain to themselves the examples and those learners appeared to learn more effectively than those who did not exhibit this behaviour. Self-explanations, formulating the unwritten steps of an example or concept, helped students understand examples and problems. Chi [45] suggested that the self-explanation effect was actually a dual process, one involving the generation of inferences and the other is repairing the learner's own mental model. If there is a divergence between the learner's own mental representation and the model conveyed by the text passage or example solution, the learner will update his own mental model.

Atkinson et al. [29] claimed that self-explanations were an important learning activity during the study of worked examples. Unfortunately, according to Renkl's research results[46], the majority of learners' self-explanation occurred in a passive or superficial way, they spent very little time studying the examples and missed the opportunities to self-explain. The minority of successful learners, however, seemed to use different self-explanation styles, for example, principle-based explanations, example comparisons or anticipative reasoning. To assist this, the learners should be guided to actively self-explain worked examples.

Chi [47] also found that sometimes students' self-explanation can lead to misconceptions, where incorrect causations were constructed. To avoid this, completed worked examples, which contain the explanations of the reason for every step, are desired and required. This ensures novices gain the right explanation through the worked examples and is the key point to help them to construct schemata.

According to Renkl et al.[48], direct training in self-explaining appeared to be effective, as are structural manipulations of examples such as adding sub goal labels, utilizing an integrated format(e.g. integrating text and diagram or aural and visual information), or using "incomplete" examples. Therefore, Roy and Chi [49] suggested self-explanation as a trainable learning strategy needs to be

considered when designing worked examples. Specifically, learners are encouraged to bring their prior knowledge to bear on the interpretation of materials and test their evolving understanding.

Therefore, worked examples should encourage learners to undertake self-explanation activities, in a way that avoids their skipping over the partially-complete sections. They should be designed to assist learners to think inwardly and reflectively. Self-explanations for learners, especially for novices, should be scaffolded at different levels of skill acquisition and schemata construction, in order to support a greater variety of learners. Faded worked examples represent a formalisation of the self-explanation principle, and are described next.

2.5.2 Faded Worked Examples

After learners have gained enough schemata, the way of prompting self-explanation in the completed worked examples, by asking for explanations of worked out steps, will become redundant. So less scaffolding for prompting self-explanation is required. “Incomplete” worked examples can be used, with the aim to help students apply what they have learned before.

Renkl et al. [50] termed worked examples with the insertion of “blanks” into the solution steps as *faded* worked examples. In this study, it was shown that incomplete examples could improve the quality of self-explanations and, as a consequence, the transfer of solution methods. Hence, they suggested that novices should firstly study fully worked-out examples, then complete steps in problems with “blanks”, and finally solve problems without any instructional support.

Renkl and Atkinson [18] discussed the reason why worked examples and faded worked examples could work effectively from the cognitive load perspective. A typical example of problem fading is when the worked-out steps in a worked example are gradually turned into standard problem solving steps by removing the worked-out elements in the steps (the explanations). Because the assistance of giving the steps is faded, the steps (explanations) are omitted. The students must generate these steps (or explanations) by themselves. Renkl and Atkinson

[18] also claimed that active self-explaining is crucial for learners in the beginning of the intermediate stage, because learners are going to learn the rationale of how to apply their basic knowledge of the domain that they have gained in the early stage. They identified self-explanation activities, which are:

1. Anticipative Reasoning: A learner tries to anticipate the next solution step and then confirms his prediction by looking it up. This activity can support learner to construct mental rules for problem solving and to check understanding.
2. Noticing coherence: A learner notices similarities or differences between different examples. Through an activity of comparing examples, it can foster the learner to induce abstract schemas.

Self-explanation refers to a learner's effort in gaining an understanding of a solution rationale, which can be considered as Germane Cognitive Load. It also can be understood as explanations provided by learners and mainly for their own benefit. These explanations were intentionally not given in the learning materials and they refer to solution steps and the reasons for them.

Faded worked-out examples should be employed at this stage to promote learners' skill acquisition in order to structure the transition from example study to problem solving. In other words, it means at this phase learners will use self-explanation to convince themselves in order to solve the problem. Hence, in the intermediate stage the learner forges links within his or her domain knowledge by using self-explanation in an effort to generalize over surface structure in order to achieve heightened understanding.

Renkl [5] also claimed that novices have limited prior knowledge and are unable to use domain specific strategies in the initial stages of cognitive skill acquisition. With this limitation, novices often rely on a means-end approach, which focuses on just trying to obtain a solution to a problem rather than understanding the material. This behaviour is immediately recognisable to the tutor of any novice programmer given a hard problem to solve. The transition from initial stages of skill acquisition to deeper understanding can be facilitated by faded examples. In the later stage of skill acquisitions, studying worked examples becomes less effective because the learners have to think for themselves, which is best

accomplished by actual problem solving. This discussion matches Frize and Frasson's idea [51], which is that different levels of learners, should be taught in different pedagogical approaches.

Schwonke et al. [52] claimed that faded worked out examples are particularly helpful for the intermediate stage of skill acquisition in which the primary instructional aim is to gain understanding and close knowledge gaps. Thereby, more of the learners' limited processing capacity (working memory) can be devoted to understanding the domain principles and their application in problem solving, especially when worked-out examples are combined with self-explanation prompts and corrective feedback. Based on their experimental results, they found that fading the worked-out steps can lead to a deeper conceptual understanding than problem solving alone, it leads a better transfer performance in learning, and meanwhile, it can reduce the learning time comparing with only example-based learning and make the learning more efficient.

Using faded worked examples makes the "dialogue" between students and learning objects happen. It has been evaluated that learning objects, which provide feedback when students interact with them, could benefit their learning progress. From the CLT angle, it can be explained that more of the learners' working memory could be used to understand the domain principles which could help them to alter their long term memory to finish the learning; meanwhile, from the education feedback angle, it can be explained that the "dialogue" between students and learning objects could ideally help students to arrive at the same level of concept engagement as the teacher's expectation.

Having identified the qualities of both worked, and faded worked, examples, guidance for designing effective examples will now be reviewed.

2.5.3 Guidance for Design Worked Example and Faded Worked Example

While worked examples have significant advantages; they do not guarantee effective learning when used as a learning methodology. Therefore, designing and presenting worked examples is not a simple task. The representations should

follow the learners' schemata acquisition processes. For example, if learning a procedure, it is better using dynamic step-to-step guidance to explain the whole process of problem-solving rather than using static methods. Meanwhile, the step-to-step guidance should help learners to recognize the previous schema in their long-term memory, and then let them abstract relevant detail based on their previous experience of solving similar problems, and also help them to map the target problem. Then the target problem can be solved based on previous experience.

Hence, Renkl [5] suggested several principles to design completed worked examples in order to guarantee that novices will gain a deep understanding, and develop appropriate schemata, when they receive worked examples in initial cognitive skill acquisitions. These guidelines can be used to create worked examples in a computer-based learning environment, which include:

- The Guideline of Self-Explanation Elicitation: Prompt or train self-explaining examples, including principle-based explanations, example comparisons and anticipative reasoning. This is regarded as central because the processing of the examples corresponds to the central knowledge-building activities, which represent desirable germane cognitive load.
- The Help Guideline: Help in form of instructional explanations should be focused on principles, be minimalist and integrated, related the explanation to the example at hand, and make the connection between the example and the explanation obvious. This sensibly supplements self-explanation activities.
- The Easy-Mapping Guideline: Design examples so that the relations between different representations can be easily detected. Extraneous cognitive load is therefore minimised.
- The Structure-Emphasizing Guideline: Make the relevant structural features salient, so the learner can detect the problem types and select the correct solution procedure. Their schemata are thereby strengthened. In order to assure the intended effects, learners should be asked to self-explain the differences and commonalities of the presented examples.

- The Meaningful-Building Block Guideline: Facilitate the isolation of meaningful building blocks in worked out procedures, which can be applied to new problem structures. This assists with the development of the solution procedures associated with the schemata being formed. Sub-goal labelling [53] is a well-researched implementation of this guideline.

Mayer [6] also suggested several guidelines to create effective and elaborated worked examples, listed below:

1. Provide explanations of each step, the explanation should be useful in helping learners understand the solution method and point out when and how to apply the solution method to solve similar problems;
2. Ask learners to generate explanations, which are self-explanations, of each step;
3. Sequence the examples by backward fading, which is starting with a completed worked example, followed by a problem with all steps worked out except the last one, followed by a problem with all steps worked out except the last two, and so on until a problem without any step worked out;
4. Ask learners to compare several examples, this aims to prompt students to make comparisons between structurally similar examples and to abstract the solution strategies from examples. This directly influences the development of the necessary abstract problem-solving schemata.

These guidelines are useful techniques for designing effective worked examples. When designing worked examples, they must be applied and embedded into the instructional design. Based on the theory of cognitive apprenticeship and worked example research finding, they both suggest that learners need to see the complex practices in action first, and see them repeatedly, in a gentle progression towards taking on the challenges of solving problems independently.

Faded worked examples could also benefit learning, so Renkl and Atkinson [7] suggested several essential elements to design effective faded worked examples. These elements are listed below:

- Sequence of isomorphic examples/problems. For instance, fading different parts of a complete worked example as several different faded

worked examples, and then offering these faded worked examples before solving the target problem.

- Prompts for principle-based explanations at worked-out step. For instance, pointing out the principle applied within such a step.
- Feedback on principle-based explanations. For instance, explaining relations between elements of principle and elements of the example.
- Fading worked-out steps for anticipations. For instance, asking questions which can lead learners to think about what may happen in the coming step.
- Feedback about anticipations. For instance, offering the right answer to the question immediately after the question step.
- In the case of wrong anticipations, providing principle-based instructional explanations. For instance, offering the explanations for the wrong answers to the question.

Based on the discussion above, it is clear to see that worked examples are suited for novices and faded worked examples are better for intermediates. Worked examples are powerful instructional tools for delivering the experience of problem solving.

2.6 Need for Interactive Worked Examples

It has been shown that worked examples are an effective means of supporting the cognitive apprenticeship necessary for developing expertise in the computing science problem domains identified. Furthermore, guidelines for developing effective worked examples resulting from a wide range of research studies have been noted.

However, outstanding observations, drawn primarily from the literature on Mastery learning[54], are that:

- a. Large numbers of worked examples are required to ensure that all learners can develop the necessary schemata for expert problem-solving.
- b. The worked examples should cover a broad range of experience levels, so that students are neither bored nor overwhelmed, quickly finding their

own level and moving through the worked examples at an appropriate rate.

The major issue in this dissertation concerns how these two points can be addressed.

The use of computer-based learning environments is proposed as a means of doing so. Appropriately designed computer-based learning environments support instructional designs required in cognitive apprenticeship, such as modelling, coaching, and scaffolding (fading systems). In such computer-based learning environments, modelling the expert's handling of complex tasks can be achieved through experts describing their own cognitive process; the coaching can be delivered through an expert's explanations; the scaffolding can be removed as the students' learning stages upgrade. Designing this kind of learning environment could help students to observe how experts apply their schemata or skills to solving problems. The process of progressively removing the scaffolding (by providing less complete explanations) encourages novices to develop their self-explanation skills. Thus while progressing from fully worked out to faded worked examples, the novice is moving from passive to active learning modes.

The limitations of worked examples from textbooks were discussed in section 2.4.1.2. They can be summarized as no interactivity for students, non-customizable by individual teachers, not enough detailed explanations for students' reviewing, and may not be suitable for student individual learning.

Furthermore, as Hundhausen et al. [55] reported, simple visualization is not enough to support successful learning of dynamic algorithmic processes. Students might only look at dynamic visualizations without understanding the context or deeper meaning. Based on his meta-analysis, he suggested that students learned better when they were engaged with a tool in some kinds of activities, such as responding to questions, making predictions or doing some experiments.

By using a computer-based learning environment, several of the above limitations can be addressed. For example:

- Interactivity is added in, as viewing is under learner control, it actively involves the student in working with the material. Students can work through the material in their own pace and can move backwards and forwards as required.
- Fully worked-out procedures can be demonstrated rather than providing only a few snapshots of the transformation. All the detailed transformational steps can be shown, as space/time is not limited.
- Full explanation of each step also can also be given, for the same reason.
- Review can be carried out at any time, as the whole procedure of developing solutions is available, unlike examples presented in lectures or tutorial.
- Faded worked examples can be well-supported, so students can actively engage rather than passively receive. Gradually fading worked-out steps of the solution can be adapted more easily by computer than on paper. Anticipating the next solution step and then confirming the prediction can also be achieved through embedded questions.

The use of computer-based interactive worked examples is proposed as a means of bridging the gap, as students are actively involved in working with the material. If suitably constructed, explanations can be provided between steps which make the “dialogue” between students and the interactive worked examples become possible.

A successful example of such interactive worked examples is Animated Database Courseware (ADbC) [56], a screenshot from which is shown in Figure 2.4. ADbC is an online database teaching tool produced by Kennesaw State University, funded by the National Science Foundation (NSF). It consists of a set of interactive software modules, designed to aid students to learn the fundamentals of database concepts. It uses hundreds of animations as worked examples to demonstrate the concepts in database systems. The interactive worked examples are not only used in the classroom to provide additional means for demonstrating concepts, but also are used for in-class and out-of-class assignments. Murray and Guimaraes [35] evaluated the use of interactive worked examples in the teaching of database concepts using ADbC and proved that worked examples do support the teaching and learning process. They got very positive evaluation results based on students’ feedback and suggested that

providing students with access to interactive worked examples increases student motivation, helps students develop deeper understanding and achieve higher levels of learning. This demonstrates that worked-examples in an interactive environment are potentially beneficial for learning.



Animated DataBase Courseware

Interactive Approach for Teaching the Principles of DataBase Concepts

DATABASE DESIGN
SQL
TRANSACTIONS
SECURITY
INFO

ER NOTATIONS
SCENARIO TO ER
ER TO TABLES
DEPENDENCIES
NORMALIZATION
DENORMALIZATION
ANOMALIES

[Introduction](#)

[Scenarios](#)

- 1:N Binary
- 1:N Unary
- 1:1 Employee
- 1:1 Faculty
- N:N Binary
- N:N Unary

[Full Screen](#)

[Help](#)

[Audio Help](#)

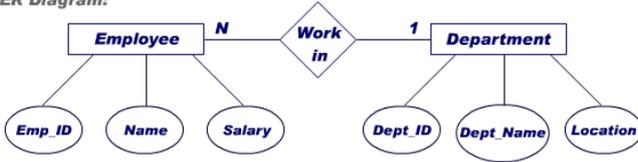
Module = E-R Diagram to Table 1:N Relation

Reload This example 1:N Exercise

Scenario: (1:N Relation)

A company is structured in a manner that an Employee may be employed by a Department (or no department at all), and a department may employ many employees.

ER Diagram:



Emp_ID	Name	Salary
100011	John Smith	30000
100022	Earl Tilley	45000
100033	Mary Black	75000
100044	Andy Wallace	55000

Dept_ID	Dept_Name	Location
001	Acct	bldg2
002	HR	bldg1

**John Smith, Mary Black and Andy Wallace work in Accounting (Acct)
Earl Tilley Works in Human Resources(HR)**

■ Show / Hide Association
▶ Proceed to Exercise

Figure 2.4 ADbC Online Teaching Tool [56]

ADbC shows an expert's problem-solving model for the learner to study and emulate, therefore novices can use it as guidance to solve problems. However, while ADbC does provide some hints about the associations between entities, these hints do not explain the reasons for the expert's decision-making judgements.

2.7 Challenges of Interactive Worked Examples

ADbC demonstrates the potential value of interactive worked examples; however, there are challenges to produce them for the typical class environment as will be discussed in this section. Two particular issues are the ease of production and the ease of customization.

2.7.1 Productivity and Customization in Learning Objects

Typical e-learning objects are one-off examples with a fixed sequence of processes, developed externally at high cost for a particular teaching context.

This means if the students' expectations rise, or the learning object does not quite fit the education sequence they have followed, it is hard to change the examples. This is because such changes would require reprogramming and recreating examples which are costly for a developer and probably impossible for a teacher with little or no programming expertise. The issue is how easily e-learning objects can be created (productivity) and how easily can they be evolved (customization).

Productivity is one common problem of creating current existing learning objects. Productivity is defined as achieving higher quality and more effective learning in affordable and acceptable ways. Technology has the potential to increase teachers' and learners' productivity. For example, it can help design software that reduces teachers' time and effort to create learning objects and speeds up accessing learning objects for learners and benefits learning outcomes. It can be understood as an increase in the value of output per unit of resource input. In the context of education, the value of the output is usually thought of as the level of achievement of the learner (sometimes the value added by the institution), and the number of learners achieving at that level. The main unit of resource that concerns us is time: teacher time and learner time. [57]

Based on this definition, the productivity of typical e-learning objects can only be acceptable when the high initial cost of creation is amortised across a large number of learners using the same object.

Customization is defined as allowing the teacher to develop their own, or to evolve existing, learning objects to fit their teaching targets. This means that the learning objects should be flexible and inclusive enough for each individual teacher to use or to modify.

Valentine's survey [58] classified papers on topics related to CS1 and CS2 presented at the SIGCSE Technical Symposium conferences between 1984 and 2003. 22% of these papers (99 out of 444) presented software tools designed to aid teachers and/or students. In spite of this large body of research and development work, only a small number of tools were used outside their home institutions. Very few tools were widely adopted, even though most important

tools can now be downloaded from websites. Pears et al. [59] claimed two possible reasons in their survey. First, tool research projects often originate in a desire to solve a local problem, either in a specific institution or a specific course. As a result, multiple tools were developed for similar purposes, like using algorithm animation tools to demonstrate the execution of code by a computer. Furthermore, local differences in instruction made it difficult for other teachers to adapt tools to their specific needs and support for tool modification was rare. Second, tools were often developed as research prototypes, and the usability of these tools was in the laboratory level, which was not good enough for realistic education contexts.

In summary, the productivity and customization involved in the current creation and evolution methods for e-learning objects are a major challenge to widespread use of e-learning objects.

2.7.2 TPACK Framework

As noted above, building effective interactive learning objects is expensive. It requires subject experts, computer graphics experts, programmers and instructional design experts working together. For example, Murray reported that ADbC required an 11 person development team working from 2007 to 2011 to build [56]. The researcher contacted Murray and asked for further details of developing time for ADbC, her reply stated that “ADbC was developed completely by students (undergraduates and Master level students) at my university. Some modules were done as class projects, others as paid student assistants. The length of time to develop any individual module, therefore, varied greatly - mostly depending on the skill level of the student developer. In general though, we have produced each sub-module in about one semester (16 weeks).” [60] In order to produce such interactive worked examples in a digital approach, much time is spent. Meanwhile, facilitating pedagogical experts and computer scientists (i.e. content experts) need to reshape each other’s models in order to design a successful digital learning object.

Mishra and Koehler [61] introduced a framework called Technological Pedagogical Content Knowledge (TPACK), based on the idea of pedagogical content knowledge (PCK) first introduced by Lee Shulman in 1986 [62], extended

with the inclusion of technology. The TPACK framework was designed to understand and describe the kinds of knowledge needed by a teacher for effective pedagogical practice in a technology enhanced learning environment. The TPACK framework describes the relationship between pedagogical knowledge, content knowledge and technological knowledge and how teachers could integrate technology in their teaching to enhance the quality of their teaching. It argues that effective technology integration for teaching specific content or subject matter requires understanding and negotiating the relationships between these three components: *Technology*, *Pedagogy*, and *Content*. A teacher capable of negotiating these relationships represents a form of expertise different from, and (perhaps) broader than, the knowledge of a disciplinary expert (say a scientist or a musician or sociologist), a technology expert (a software developer) or an expert at teaching/pedagogy (an experienced educator).

Harris et al. [63] adapted TPACK and added the knowledge components inside the framework. This is shown in Figure 2.5. They claim it is important that TPACK based professional development for teachers is flexible and inclusive enough to accommodate the full range of teaching philosophies, styles and approaches. Hence, they proposed a new approach based on the TPACK framework that can help teachers successfully integrate technology into their practice. This approach goes beyond technological strategies and emphasizes the importance of helping teachers develop and apply integrated and interdependent understandings of technology, pedagogy, content and context.

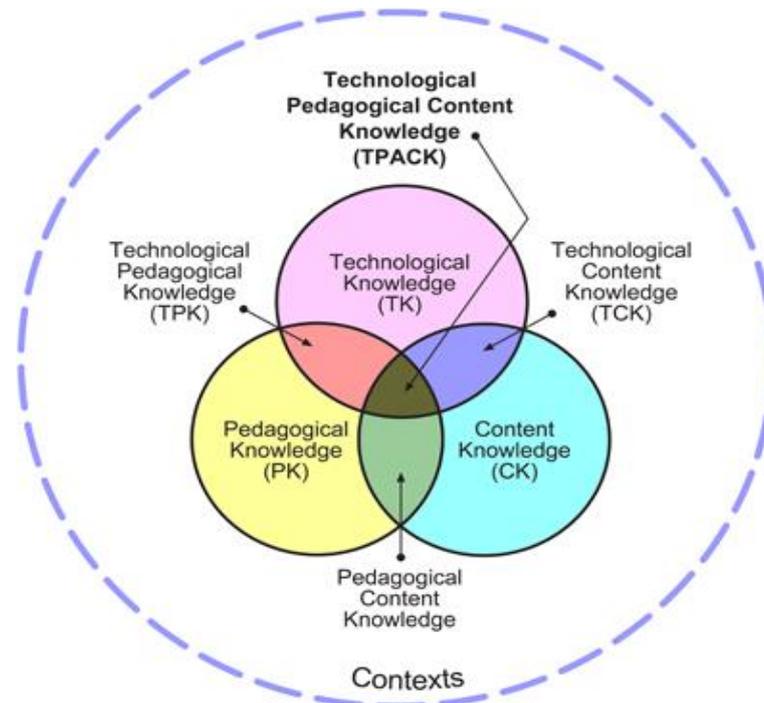


Figure 2.5 TPACK Framework Diagram [63]

Archambault and Crippen [64] highlighted complex relationships that exist between content, pedagogy and technology knowledge areas and pointed out that the TPACK may be a useful organizational structure for defining what it was that teachers need to know to integrate technology effectively. They examined teachers and measured their knowledge, which included pedagogy, content, technology and the combination of each of these areas. The results indicated that teachers felt good about their knowledge related domains and were less confident when it came to technology. Correlations among each of the domains within the TPACK framework revealed a small relationship between the domains of technology and pedagogy, as well as technology and content. However, there was a large correlation between pedagogy and content. Hence, lowering the threshold for technology entry for teachers is clearly necessary for increased E-Learning. It is the key point for making E-Learning become possible and available to most of the teachers.

Given the key importance of lowering the technology threshold, so that teachers can create computer-based activities, the next section explores authoring environments and tools as a way to remove or reduce the technology factor in the TPACK model.

2.8 Authoring Environments and Tools

The technology barrier can be significantly reduced by removing the need for teachers to use programming skills or to have to work with third party experts to develop interactive worked examples. In other contexts, an authoring environment can be used to achieve this end. An authoring environment is a toolkit that enables digital artefacts or content to be created without programming skills. For example, the Microsoft Word Application is a typical authoring environment for producing electronic documents. In considering authoring environments, two particular aspects are relevant in this context: first the authoring environment should be usable by the intended population of authors, and second, the artefacts developed using the environment should be appropriate for the intended audience, in terms of usability and functionality.

In the educational context, these two populations are teachers and learners, respectively. The needs of the learners have already been described in detail and summarized in the earlier section 2.5.3, outlining guidance for creating best-practice worked examples. However, these guidelines are independent of context, so that they could refer to spoken, paper-based or computer-based worked examples. The issue involved in the computer-based context are now addressed, for each group in turn.

2.8.1 Empowering the Teacher

An educational authoring environment provides teachers with a visual authoring environment for creating artefacts like web sites, interactive hypermedia, microworlds, or simulations. Locatis and Al-Nuaim [65] gave a useful definition of an authoring system: “the term authoring tool refers to a range of software products having utilities for composing, editing, assembling, and managing multimedia objects, while the term authoring system refers to a subset of these products allowing multimedia development without programming.”

Locatis and Al-Nuaim [65], Bodendorf et al. [66], Recker et al. [67] and Hsiao et al. [68] argued for the need to provide authoring tools for teachers to create rudimentary worked examples without special training. Authoring tools provide a non-programming environment where authors are prompted at every step as

they enter lesson content and specify instructional strategy. Once entered, the system automatically produces the corresponding error-free code, which controls the lesson's presentation. Hence, authoring tools can reduce the technological knowledge effects in the TPACK framework. Using authoring tools eases development and can help teachers to bring control of the authoring process back to the content expert. Tools having authoring utilities specifically for instruction can guide users in developing more effective products. However, Avner et al. [69] stated that the authoring system can improve quality, but not guarantee it. They may make it easier to author poor instruction as well as good, because anyone can be an author. Hence, the author should at least have good content and pedagogy knowledge in order to use an authoring environment to produce good worked examples.

Some authoring environments were developed in the past. However, they were designed for different purposes. CTAT[70], REDEEM [71], ASTUS [72] and EON [73] are authoring environments for creating Intelligent Tutoring Systems. By authoring ITSs to engage between the teacher's schema and the students' schema, the teacher has to model students' behaviour through using feedback messages to correct students' schema or using hints to guide students to apply the right schema to finish the exercises. MatrixPro [74], RIDES [75], SimQuest [76] are authoring environments for creating computer simulations embedded in an instructional environment to support discovery learning and guided practice. QuizJET [68] is an authoring tool for teacher to create on-line quizzes and questions, which are used by students to do self-assessment. Microsoft FrontPage, Adobe Captivate 6 [77], Course Lab[78] and GLO Maker[79] are authoring tools for publishing E-learning contents, which can be used for students learning by themselves as an alternative to attending lectures. Unfortunately, none of these authoring environments focus on creating interactive worked examples, which can transfer an expert's schema and present solutions in a step-by-step fashion.

Microsoft PowerPoint is an authoring tool designed for creating slideshows to give presentations. As such, it plays a significant role in helping teachers to develop lecture content in the current education environment. The slides are usually linear and may contain hyperlinks for jumping between different sections.

This allows the user to get more detail on a subject of interest. In addition, it allows the user to add audio and video clips in a presentation. In a single slide it allows the user to define a sequence of showing different pieces to form an animation. Basuhail [80] identified one of the reasons for using the Microsoft Office package for designing, implementing, and delivering worked examples was that it is built on the users' experience of using PC software. Hence, teachers use it to prepare worked examples, which can be demonstrated inside the lecture with oral explanations in a step-by-step fashion. In the lecture, teachers also can ask questions in order to make students critically think about the worked examples. But after the lecture, when students reviewed these worked examples, as the oral explanations were missing, the teacher's thinking process for solving problems could not be revealed. Using PowerPoint slides in the lecture, presenting concepts in sequence one after the other, the teacher can manually refer back to previous written content (sometimes to information recently presented, and other times to the content originating much earlier) in order to explain how the concepts are applied when solving problems. Unfortunately, when students use these slides after the lecture, they do not have the same ability as the teacher had to jump between different slides, so these worked examples become quite similar to the worked examples in a textbook. This is a root problem of using PowerPoint to develop worked examples, as it is designed for giving presentations. Hence, the power of the worked examples developed using PowerPoint is very limited outside the lecture context.

By using authoring tools, teachers who have the pedagogy and content knowledge only can produce interactive worked examples. The area of technology knowledge in TPACK framework will shrink, so the teacher only needs to pay attention to the trends in learning, not in technology. Technology knowledge will not be a bottle neck for individual teacher to deliver personalised interactive worked examples.

2.8.2 The Essential Features of the Teacher User Interface

An authoring environment for interactive worked examples will need to allow the teacher to:

- Describe the thinking process of solving problem following the cognitive apprenticeship model;
- Explain the reason for decision-making in a step-by-step fashion while the worked example is demonstrated;
- Embed questions between steps in order to encourage students to critically think about the worked examples;
- Identify the applied concepts in order to reveal the relationship between the problem and the solution.

2.8.3 The Essential Features of the Student User Interface

Freitas and Neumann [81] stated that the user interface is the key point for an effective teaching tool. It should be intuitive and closely follow the requirements of the individual learners and support personalized learning. So the learners should control their individual learning pace, choosing when and where to use the tool and the pace of learning.

As discussed earlier, self-explanation plays an important role in the worked example assisted learning process. The provided worked examples should be able to encourage students to generate self-explanation while processing problems. Hence, an essential feature of student interface is to allow and encourage students self-explaining. In a computer-based context, this can be achieved through asking students questions, and letting them type in explanations of what they have done or what they need to do next.

Jeung et al. [82] suggested that highlighting the relevant contents of worked examples, linked to the explanation of this part of the contents, can reduce the cognitive load. This allows students to devote their cognitive resource to understanding the worked example, which leads to enhanced learning. Hence, allowing students to see the highlighted contents with the explanation together is another essential feature of the student user interface.

To summarize, the essential features for student user interface are allowing students to:

- Control the speed of viewing worked examples;

- See the highlighted contents in worked examples together with the linked explanations.
- Answer questions and access model answers.

2.9 Summary of the Research Context

A lot of ground has been covered so far in this chapter, so a summary is appropriate here. The research context is in computing science and the problems to be solved are often transformation-based tasks, which include generating abstractions and making decisions. However, the transformation process does not follow a formulaic thinking process. There are guidelines to aid the transformation process, but previous successful experience plays a crucial role during the process of problem-solving, especially when making decisions. Hence, scaffolding students to develop successful problem-solving experience is essential and important.

However, the current instructional design model offers very limited opportunity for students to obtain experience of problem-solving directly from the teacher. Worked examples are important for building experience, but students are not exposed to enough suitable worked examples before they are expected to solve problems. Meanwhile, offering only a few static worked examples to students with a broad range of experience levels can lead them to feel either bored or overwhelmed. Furthermore, due to the feedback issue, the teacher has very little data about students' experience of using worked examples. Therefore, teachers can have only a poor sense of their students' learning progress.

Interactive worked examples facilitate students' use of worked examples on their own, since a number of the limitations of static worked examples are overcome. ADbC is a good example, but to build these one-off examples required a large amount of programming effort and they cannot easily be modified by practitioners.

Hence, the aim of this research is to offer an authoring environment for teachers to easily create usable interactive worked examples to be used by students and allow the teacher to modify the examples based on feedback from the students' use.

2.10 The Proposed Research Work

The aim of this research is to show that a prototype tool, known as IWE, can be created that allows a teacher to create worked examples for students to use interactively in their own time. The types of worked examples that will be supported are those that present transformations between different representations of a problem, revealing the teacher's problem-solving approach to the student. Teachers must be able to create these worked examples without the need for bespoke programming. The cognitive apprenticeship model has a major impact on the way of presenting worked examples in this research project. It also provides strong guidance to the design and implementation of the teaching system.

2.10.1 The Concept of IWE

Figure 2.6 shows the overall concept diagram of IWE. There are two distinct types of user, teacher and student, and so two separate user interfaces to the system are required. For the teacher, the primary purpose of the system is to be able to create and modify interactive worked examples for use by the students. For the student, the primary purpose of the system is to be able to interact with the teacher's worked examples and explanations to improve their subject knowledge. Adding a feedback mechanism for the student allows them to send questions or comments asynchronously to the teacher via the system. The advantage of sending feedback through IWE, rather than sending an e-mail directly to the teacher, is that the feedback is linked to a specific worked example. This means that a number of feedbacks for a given worked example can be presented to the teacher in a structured way. Collected feedback from the students is returned to the teacher, together with usage data and a summary of answers to embedded questions collected by the tool, allowing them to modify interactive worked examples. This might be required to correct errors, extend worked examples or improve explanations. Some student feedback may be dealt with by individual contact between the teacher and student as appropriate.

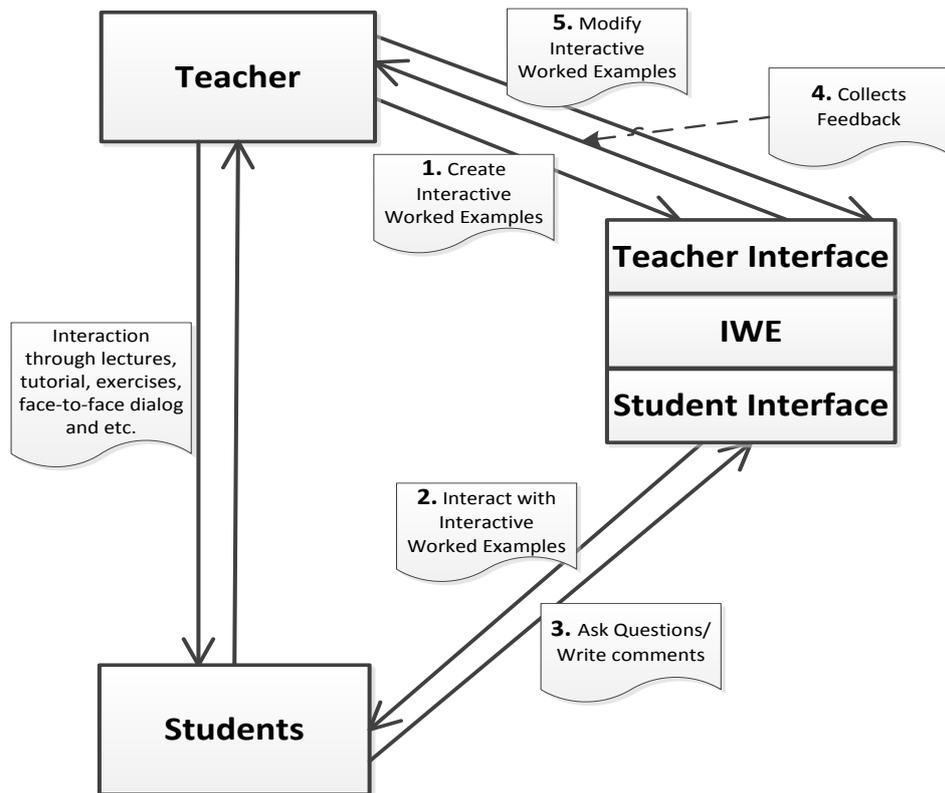


Figure 2.6 Concept Diagram of IWE

The concept diagram of IWE shown in Figure 2.6 can be summarized as a loop with 5 steps.

1. A teacher creates worked examples by using the teacher interface of IWE, which delivers them to students;
2. Students access and interact with these delivered worked examples using the student interface of IWE. While students are interacting with IWE their actions can be logged by the system and their answers to embedded questions recorded;
3. Students send feedback, such as questions, during the interaction or comments about the worked examples, back to IWE where the teacher can access them;
4. The teacher collects: students' questions and comments, information about student usage of the tool, and answers to embedded questions (in faded worked examples), and analyses this data.
5. The teacher may modify or redesign the worked examples based on their interpretation, or even could give more explanations of major questions in a future lecture or tutorial.

Hence, this new authoring environment is not designed as a complete self-teaching and stand-alone system; it will be used within the current teaching and learning framework in order to meet different teaching and learning styles for both teachers and students. For teachers it can speed up building and modifying interactive worked examples and make it easier for teachers to produce them. Students can individually use these interactive worked examples with embedded explanations, to learn by themselves anywhere and anytime on demand. They can provide feedback to teachers who can use it, together with usage data collected by the system, to improve the on-going teaching process.

2.10.2 The IWE Model

To reiterate, a worked example is a demonstration of the process used to solve a problem. The author determines how to present the problem solving process, deciding on the number of steps to present to the student, and the explanation required at each step to understand what is happening. Given that this work is concerned with transformation-based problems, a starting point is the problem definition and a complete solution. The solution process typically involves identifying information in the original problem and incrementally revealing the part of the solution derived from this information. Fundamental aspects of an interactive worked example for a transformation-based problem are therefore:

- A problem and a solution, either of which can be graphical or textual documents (e.g. textual for a problem statement and graphical for an ER diagram solution.)
- A sequence of steps that break the problem solving process down into manageable chunks for the target audience. Each step has the following:
 - An explanation of the problem solver's thinking process at this point, and also an explanation of the transformational activity, if any, in this step.
 - Optionally, a transformation that draws on information in the problem document and maps it to a particular part of the solution. To match the process in real life, the solution will best appear to the learner incrementally, as each step is addressed.
- Asking questions and giving model answers could also be embedded into the demonstration of problem-solving process.

To illustrate how this model could be realised, some typical steps in an interactive worked example for the problem solution described in 2.1.2 can be used. The main display could consist of three panels, one for the document containing the problem description, one for the ER diagram that is the solution and a third panel for the explanation of each step. Following the expert's approach there could be another panel for recording potential attributes, which would eventually show the final set of attributes for each entity. However, for simplicity it will be assumed that the attributes will eventually be added the ER diagram in this interactive worked example.

After some general introduction to the IWE and its use, the first few steps might be:

1. present the problem description given at the beginning of 2.1.2; the explanation could link this to the relevant course and identify the objectives of this worked example;
2. highlight the last two sentences of the problem description and the need to be clear about what is required as a solution to this problem;
3. remind the student what an entity is and suggest that they scan through the problem description to see if they can identify possible entities;
4. highlight 'company' in the problem description and explain why this is not an entity;
5. highlight 'department', explain why it is an entity and reveal the entity Department in the ER diagram.

Moving on, 'project' and 'employee' would be handled in a similar way to 'department' and the ER diagram would consist of three entities. In order to increase student engagement with the example, step 3 might be followed with a faded step to engage the student's anticipative reasoning by asking how many possible entities they can identify.

This limited example shows one of the most fundamental requirements of IWE: the need for very detailed structure. In text documents it is necessary to be able to highlight or reveal individual words or phrases and in graphical documents each component needs to be treated as an object to be highlighted or revealed also.

2.10.3 IWE Requirements

The requirements for an interactive worked examples authoring and presentation tool need to address the activities of the teacher author of worked examples and the student viewer of those examples. The needs of the student viewer will be explored first, as these drive the whole exercise.

The requirements of the student interface, given below, deliver worked examples building on the best practices identified in Section 2.5.3.

Requirements for the Student Interface are to allow students to:

1. interactively explore worked examples. The interface should:
 - a. present interactive worked examples, step-by-step with integrated explanations of the changes within each step;
 - b. within each step, be able to highlight relevant components of the problem or solution, for example, the information from the problem that is being used in this step;
 - c. when developing a transformation, be able to reveal new parts of the developing solution.
2. answer embedded questions in faded worked examples and see model answers;
3. send feedback to lecturers.

Considering the teacher user interface, in order to present the interactive worked examples step-by-step a process structure must be defined that allows the teacher to specify a sequence of steps and the changes in each step together with the necessary explanation.

To support highlighting or revealing parts of a textual document or diagram we need textual and graphical editors that provide fine-grain control over components such as an individual word, phrase or diagram component. Widely used editors such as Microsoft Word or Visio do not allow the access to internal structures that would make this possible. Therefore, we need to provide tailored textual and graphical editors to create documents and diagrams in IWE.

Teachers need to be able to define questions and present answers in faded worked examples. The teacher also needs to be able to collect feedback from students and from IWE to inform possible evolution of the interactive worked example or other changes to teaching.

Requirements for the Teacher Interface are to support:

1. the creation of documents that represent the source and the result of a transformation-based problem. These are structured into individual elements to support the process definition described in step 2;
2. the definition of a process to drive the interactive display of a worked example. Transformation steps in the process may include actions to: reveal or hide parts of a document; highlight part of a document; or highlight related elements of different documents;
3. the creation of explanations for each transformation step between the documents;
4. the definition of embedded questions and model answers to support the development of faded worked examples;
5. access all the feedback about students, including the correctness of students' answers, log information, and students' questions or comments;
6. the modification of worked examples after analysis of feedback, including modifying documents, processes or explanations and revising questions and model answers .

These requirements are expanded in detail in chapter 4.

The other part of the background is about usability as it is essential for the evaluation of IWE. So the next chapter is going to review literature about usability.

Chapter 3 Usability Literature Review

The thesis statement claimed that IWE is a new authoring environment for teachers to create their own interactive worked examples. It is a new software application; therefore evaluating its usability is essential. IWE is not only a tool for reducing the technology bottle neck of creating interactive worked examples for the teacher, but also a tool for students using these examples to support their learning. There are two different types of users, with different objectives that should be taken into account while carrying out the evaluation process. Hence, in this chapter, usability evaluation methods will be reviewed in order to apply suitable evaluation methods at different iterative development stages. The research questions will be answered based on the results of these evaluations from teacher and student users.

3.1 Usability Definition

According to ISO 9241-11[83], usability is "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." According to Nielsen [84], "usability also refers to methods for improving ease-of-use during the design process." It is a non-functional requirement, but cannot be directly measured. However, it must be quantified by means of indirect measures, for example, the number of reported problems with ease-of-use of a system.

Nielsen [84] also mentioned that "in computer science, especially in the area of Human-Computer Interaction, usability studies the elegance and clarity with which the interaction with a computer program or a web site (web usability) is designed. Usability differs from user satisfaction and user experience because usability also considers usefulness, which is particularly important for evaluating education software."

According to this definition, the usability study of IWE should be conducted in the education context to measure the effectiveness, efficiency, and satisfaction based on the use experience of target users. Testing is essential work in software engineering. Therefore, it is necessary to test the usability of IWE during the iterative developing process.

Usability studies, which focus on the detailed measurement of the user experience of using the tool, are not the aim of this research. However, applying suitable usability testing techniques to answer the research questions is the desired goal. So the next subsection reviews the methods that have most commonly been used for usability evaluation.

3.2 General Methods of Usability Evaluation

Usability evaluation is any analysis or empirical study of the usability of a prototype or software. The goal is to provide feedback during software development, supporting an iterative development process [85]. Nielsen[84], Preece and Benyon [86] mentioned that usability evaluation can be carried out in the different lifecycle stages in software development. Evaluation methods include: expert evaluation, observation, survey evaluation with questionnaires and interviews, logging actual use and asking users for feedback. The different methods imply different types of evaluators, different numbers of users, and different types of data to be collected. A brief review of these methods is provided below.

Expert evaluation, also known as heuristic evaluation, is normally carried out by experienced people in interface design and human factors research who are asked to describe the potential problems they foresee for less experienced users. These experts often suggest solutions for the problems they identify. This method is efficient and provides prescriptive feedback, especially in the early stage of development. However, experts should not have been involved with previous versions of the prototype under evaluation and they should have suitable experience. The role of the experts needs to be clearly defined to ensure that they adopt the proper perspective when using the prototype. The tasks undertaken and the materials given to the experts should be representative of those intended for the eventual users. Finally, the form of reporting adopted by the expert needs to be specified so that information is obtained about the most important problems [84].

Observational evaluation implies collecting data that provide information about what users do when interacting with educational software. Nielsen[84] claimed that observing eventual users working with the system, was an extremely

important usability method for both task analysis and for information about the true field study. Several data collection techniques may be used, for example, video recording in order not to interfere with user and taking notes while observing the user. According to Preece and Benyon [86], two broad categories of data may be obtained: how users tackled the given tasks, where the major difficulties lie, and what can be done; and performance measures like frequency of correct task completion, task timing, and frequency of participant errors. Albert and Tedesco [87] evaluated the reliability of self-reported awareness measures using eye tracking data through asking usability participants if they noticed a particular element on a website or software application. They reported that in the usability testing, there is reliability in self-reported awareness measures. Usability practitioners should feel confident in collecting self-reported awareness measures from participants, because at least most of the time when a participant reports seeing an object they actually did.

Survey evaluation aims to assess users' opinion or to understand their preferences about an existing or potential product through the use of interviews or questionnaires. This is a useful method for studying how users use the system and what features they particularly like or dislike. From a usability perspective, questionnaires and interviews are indirect methods, since they do not study the user interface itself, but only users' opinions about the user interface. However, they are direct methods when it comes to measuring user satisfaction.[84] Valid and standard questionnaires to study the usability of the software can be used in different circumstance. For example, SUMI (The Software Usability Measurement Inventory, 50 questions) is a rigorously tested and proven method of measuring software quality from the end user's point of view[88]. Others like PSSUQ [89](The Post Study System Usability Questionnaire, composed of 19 questions), QUIS[90] (The Questionnaire for User Interaction Satisfaction, composed of 27 questions), and SUS [91] (System Usability Scale, composed of 10 questions) are designed to assess user satisfaction after participation in a scenario-based usability study.

Here a little more attention will be spent on the SUS questionnaire, as it is mature and widely used and has become an industry standard referenced in more than 1200 research publications and has probably been used in many more

evaluations that have not been published. [92] SUS was developed by Brooke in 1986, which he called a “quick and dirty” scale to satisfy the need for a questionnaire that is both short and reliable. The goal was to have a questionnaire that could be used immediately following a laboratory test of the usability of new software or hardware. SUS was intended to provide a measure of the user’s subjective view of the usability of a system, but not intended to provide diagnostic information. [91] SUS is not the only questionnaire to measure usability, but it is one of the best. It also has proven reliability and validity, explained below.

- SUS is short with very easy scale to administer to participants, only 10 questions to be answered that are enough to assess the usability. The questionnaire is shown in Appendix 13. Analysis by Lewis and Sauro [93] and confirmed by Borsci et al. [94] showed that SUS can measure two factors, which were usability and learnability (Question 4 and 10).
- SUS is a reliable and valid questionnaire, which can be used on a small sample size. Tullis and Stetson found that at least 12-14 participants were needed to get reasonably reliable results[95]. Bangor et al. reported by comparison with other questionnaires, the internal reliability of SUS was in the range between 0.89 (SUMI) and 0.96 (PSSUQ) [96].

SUS does not measure the efficiency and effectiveness directly and accurately, but it asks about users’ attitudes about efficiency, effectiveness and satisfaction. This attitude information is extremely valuable, as when users can and will tell what they think, it is the first step in improving the usability of an application[97]. For evaluating the prototype of a new software application, it is a reasonable choice.

Another particularly useful technical method is to use software logging. Normally, logging is used as a way to collect information about use of a system in the field after release, but logging can also be used as a supplementary method during user testing to collect more detailed data. This technique records the interaction between the user and the software, and automatically collects statistics about the detailed use of the system. The data is collected automatically from a large number of users working under different circumstances and unobtrusively without influencing the user’s working style. As

the data usually consists of a time-stamped log of user input and software responses, it is possible to reconstruct what the user was doing and the time spent on each feature, and also to analyse the frequency of use of certain features. However, a major problem with logging data is that it only shows what the users did but not why they did it. It is possible to combine logging with other methods, such as interviews, where users are shown data about their own use of the system and asked to elaborate on whatever interesting phenomena may be evident in the data. For example, a user who did not use a certain feature in a system might be asked why they did not use this feature. [84] The way of using logging suggested by Nielsen is through collecting statistics of low level usage data, related questions can then be prepared and asked in later interviews. However, measuring the low level of interaction is not the target of this project, the aim of using logging should focus on understanding the educational implications of IWE, for example, whether the time students spend on using the tool match the teacher's prediction or not.

User feedback is a major source of usability information, if the developer collects the user's feedback and responds to this useful information. Very often there is a tendency to get user feedback from dissatisfied users, who complain about a feature in the system while using it, or from the most vocal users. So the user feedback may not always be representative of the majority of users. It is recommended that other users should be actively sought out and observed or questioned. However, user feedback has several advantages:

1. It is initiated by the users, so it shows their immediate and pressing concerns.
2. It is an ongoing process, so feedback will be received without any special efforts to collect it.
3. It will quickly show any changes in the users' needs, circumstances, or opinions, since new feedback will be received whenever such changes occur. [84]

Other methods introduced by Rubin [98] can also be applied such as: experimental evaluation, focus group, walk-through, paper-and pencil evaluations, usability audit, field studies, and follow-up studies. Not all methods

have to be used for a single usability study, but normally, a combination of methods is selected according to the needs and constraints of a project [84].

3.3 Extra Work for the Usability Evaluation of Educational Software

Squires and Preece [99] were aware of the limitation of applying these usability evaluation methods to evaluate educational software. They argued that just because an interface was easy to use did not mean that it was designed appropriately from an educational perspective. There was an essential relationship between the two which must be addressed to ensure good educational software design. This awareness was also introduced by Nielsen [84] as the definition of usefulness, which was the issue of whether the system can be used to achieve some desired goal. Usefulness can be divided into two categories of utility and usability. The utility is the question of whether the functionality of the system in principle can do what is needed, and the usability is the question of how well users can use that functionality. For example, educational software or courseware should have utility allowing students to learn from using it. Later on, Squires and Preece [100] proposed an initial set of “learning with software” heuristics, which can be used to predict the quality of the educational software that takes account of both usability and learning issues from educator’s point of view. Three important points were:

1. When learners use the educational software to learn, the model formed by learners should be consistent with the teacher’s predictive model.
2. The education software should match with the curriculum relevance and teacher customization, as teachers will often feel the need to match particular curriculum requirements or adapt software to the specific needs of their students. A good education application should be able to facilitate teachers to do this.
3. Strategies for helping learners to recognize cognitive errors, to diagnose and recover from them should be supported by the software.

Mayes and Fowler[9], and Hornbaek [101] also claimed that the concept of usability in the field of educational software has to be adapted to pedagogical approaches and theories of learning, due to traditional usability principles not

being sufficient to guarantee successful learning. Avouris et al. [102] applied traditional usability measures to rate the usability of one of two online learning systems. Students had to perform a test after using one of the learning systems for 15 min. They found that the group of students using the system that was rated most usable showed significantly better learning outcomes. However, later on Tselios et al. [103] claimed that educational software should not simply support the efficient execution of a task. In specific cases, increased usability could have a negative impact on learning, since executing a task efficiently may prevent essential learning processes. For example, in the database area, the Aqua Data Studio [104] provides a Visual Query Builder, which allows students to construct sophisticated database queries without having to know the syntax of SQL statements.

At this point, it's worth noting that determining the usability of an application is not the same as evaluating its educational effectiveness. This can be interpreted as stating that a highly usable product is no guarantee that learners will retain information. Nonetheless, it is virtually assured that an application with low levels of usability will not enable learners to access and assimilate information at all. Hence, ensuring educational software is highly usable is the prerequisite for building effective educational software.

Hollender et al. [105] conducted a literature review about research on cognitive load theory and research on usability in human-computer interaction. Based on the result of this review, they proposed two conceptual models shown in Figure 3.1 and Figure 3.2, which integrated cognitive load theory concepts and usability concepts. These models are intended to offer implications for the design of complex educational software. The core point in these two models is that educational software should focus on assisting cognitive processes of students rather than on supporting efficient execution of specific tasks.

These two models suggest ways of conducting usability testing for educational software. By deeply understanding these two models, it points out the responsibilities of the computing researcher when designing new educational software. It also identifies the responsibilities of the teacher when using educational software.

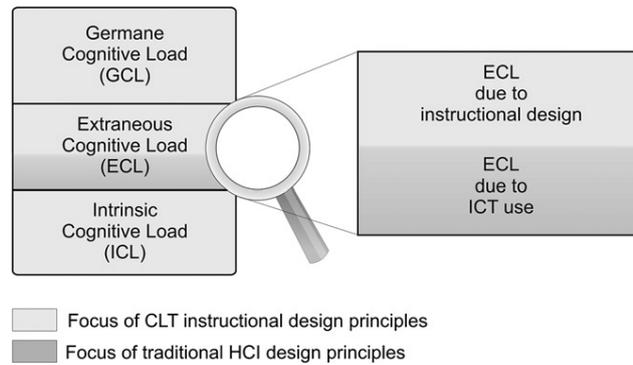


Figure 3.1. Components of CLT in Information and Communication Technology (ICT) Supported Learning [105]

The first model in Figure 3.1 outlined previous research results of CLT, which was introduced in section 2.2.2 of chapter 2. Intrinsic Cognitive Load depends on students' previous experience. Germane Cognitive Load refers to the cognitive learning process, how students process the information to construct new schema. Hence, when teachers design worked examples, the levels of Intrinsic Cognitive Load should be provided appropriately, based on understanding students' previous experiences as the starting point. The Intrinsic Cognitive Load can only be provided at a suitable level for the majority of students, not for individual student. If worked examples are too advanced for the students, the Intrinsic Cognitive Load is too heavy; because students struggle to learn from them and will not be able to generate Germane Cognitive Load for constructing new schema. If the worked examples are too simple, students may find the worked examples redundant after using them; because no new schema can be constructed after using these worked examples. This model reveals that the amount of Extraneous Cognitive Load can be influenced by the design of learning objects and the complexity of software use. If a highly usable software application can be designed or enough training can be provided to learners, the software factor in extraneous cognitive load can be minimised. As discussed before, a good user interface does not guarantee educational value, but a poor user interface is unlikely to deliver high educational value. Hence, designing an intuitive user interface for students is essential, and evaluating the usability of this user interface by applying traditional usability evaluation methods is necessary. The remaining amount of cognitive load should be taken into account by the teacher when designing the learning objects. [105]

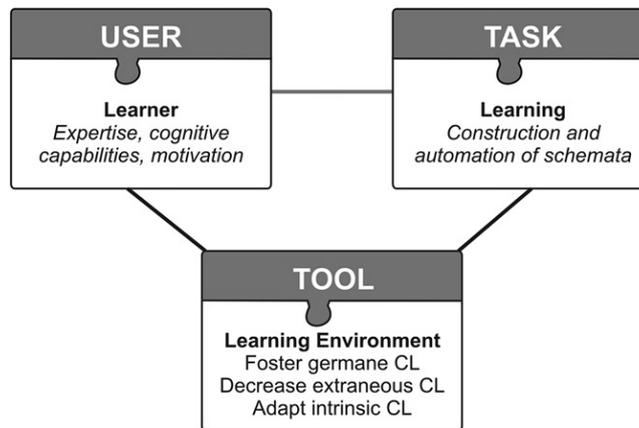


Figure 3.2. User, Task and Tool in an E-Learning Context, defined according to CLT. The goals of the tool are related to the three types of cognitive load. [105]

The second model, shown in Figure 3.2, integrated the CLT concepts into the usability concepts. It described that the relationship between the learner and the educational software. CLT defines the user as a learner and the task as learning. Learners need to finish the task though using the educational software. If learners can use the software to finish a task successfully, they should be able to construct new schema after finishing the task. The goal of educational software is to adapt the Intrinsic Cognitive Load, to reduce the Extraneous Cognitive Load and to foster the Germane Cognitive Load, through assigning tasks to users. Hence, when designing educational software, providing facilities following the best practice instructional design guidance from education research results is essential. By providing the appropriate level of worked examples as tasks, learners should be able to finish the task successfully to adapt Intrinsic Cognitive Load under the designer expectation. Decreasing the Extraneous Cognitive Load can also be achieved through reducing the irrelevant learning of using the software by making the software easy to use. When evaluating tool usability, traditional HCI evaluation methods to evaluate the learner user interface can be applied.[105]

To summarise, when initially designing educational software, attention should be paid to designing an intuitive user interface for students. This kind of easy to learn and use interface can reduce the factors which could increase Extraneous Cognitive Load for learners. When evaluating the usability of educational software, attention should be paid to whether the students can engage with the contents as the designers expect or predict. The usability of the user interface needs to be good enough to be operated by learners; otherwise they may stop using it. However, it is less important to provide a highly usable interface for

teachers, as they can tolerate a sufficiently usable user interface to achieve their objective, if they are highly motivated. Later on, attention can be paid on improving the user interface for the teacher.

Based on the discussion above, the strategy of evaluating IWE can be described below, with the aim of answering the research questions:

- For teacher users, usability evaluation should be qualitative. It should focus on proving the concept, which means the teacher user should be able to create and modify interactive worked examples by using this authoring environment. A usage experience report is required that identifies the problems during the process of creating and adapting worked examples. The teacher must be aware of expected Intrinsic Cognitive Load before creating worked examples, which means that the designed worked examples need to match the students' previous experience. The authoring environment must provide the means for the teacher to achieve instructional designs by reducing the Extraneous Cognitive Load, and fostering Germane Cognitive Load.
- For student users, usability evaluation should focus on finding the evidence from looking at the users' log traces. Students' feedback about worked examples can be another resource to identify whether the level of Intrinsic Cognitive Load is appropriate or not. All evidence aims to demonstrate delivered interactive worked examples are useable and useful to students. It also needs to measure the satisfaction with using this tool to ensure the Extraneous Cognitive Load is being minimised.
- For both teacher and student users, usability evaluation should focus on the utility of feedback, for example, how to apply the feedback to adapt the Intrinsic Cognitive Load, which needs to be processed by the students. The aim is to evaluate whether the teacher can take action to modify the worked examples based on students' feedback or create new examples dynamically for the ongoing course progression.

Chapter 4 Requirements

The discussion in the background chapter showed that a new authoring environment for creating interactive worked examples is required. The new authoring environment aims to reduce the technology overhead of producing Interactive worked examples, so that it can be used by any teacher without the need for bespoke programming. The worked examples produced should communicate concepts to students, requiring the worked examples to contain explanations when students interact with them. The authoring environment should have the ability to collect students' feedback. Teachers should be able to access these students' responses after worked examples are used and access a summary of students' answers, if faded worked examples are presented. It would be useful to provide a summary of students' actual usage to the teacher, based on log files of students' usage. The teacher should be able to use this authoring environment to modify worked examples in response to feedback.

Section 4.1 describes the general requirements, which include the structure used for presenting interactive worked examples, basic concept definitions of IWE and the data model of IWE for representing interactive worked examples. Section 4.2 describes the detailed requirements for the two types of users, teachers and students, which identify the different user interfaces to be created for these users. Section 4.3 presents the Use Case diagram of IWE and explicitly explains how the teacher and the student can access the system. Section 4.4 describes some other necessary requirements, including logging requirements and other system requirements.

4.1 General Requirements

The basic requirements of IWE are:

1. The environment should produce interactive worked examples, which must be usable by students. The students' user interface should be user-friendly with straightforward operations and should also keep students' attention on learning.
2. The environment should allow teachers to set up their own worked examples using an authoring approach and the internal relationship of

these worked examples should be revealed in a visible way. For example, a teacher has an answer to a problem that needs to be explained in their mind (it can be thought of as a big picture); it should allow the teacher to author the process of solving the problem, visualizing the whole thinking process.

3. Students are able to interact with these worked examples on demand and should be able use the worked examples to meet their individual learning purpose. For example, trying to learn some new concepts or understand concepts more deeply.
4. Teachers should obtain students' feedback about worked examples that they set up in order to improve their teaching and worked example design.
5. The environment should support modification of worked examples. These modifications are possibly based on the feedback from students, embedded question answers and data from log files.

4.1.1 The Overall Structure of IWE

Figure 4.1 shows the main components of IWE, how IWE structures them to present worked examples and how teachers and students can access them. There are two distinct types of user, teachers and students, and therefore two separate user interfaces to the system are required. The teacher interface must support the creation and modification of interactive worked examples, in the teacher's mind, from a textbook or from a teacher's handout, or past examination questions. The teacher needs to be able to define the documents used for representing the worked example and define the process, including explanations, which will display the interactive worked example to the student. The student interface must support the exploration of interactive worked examples.

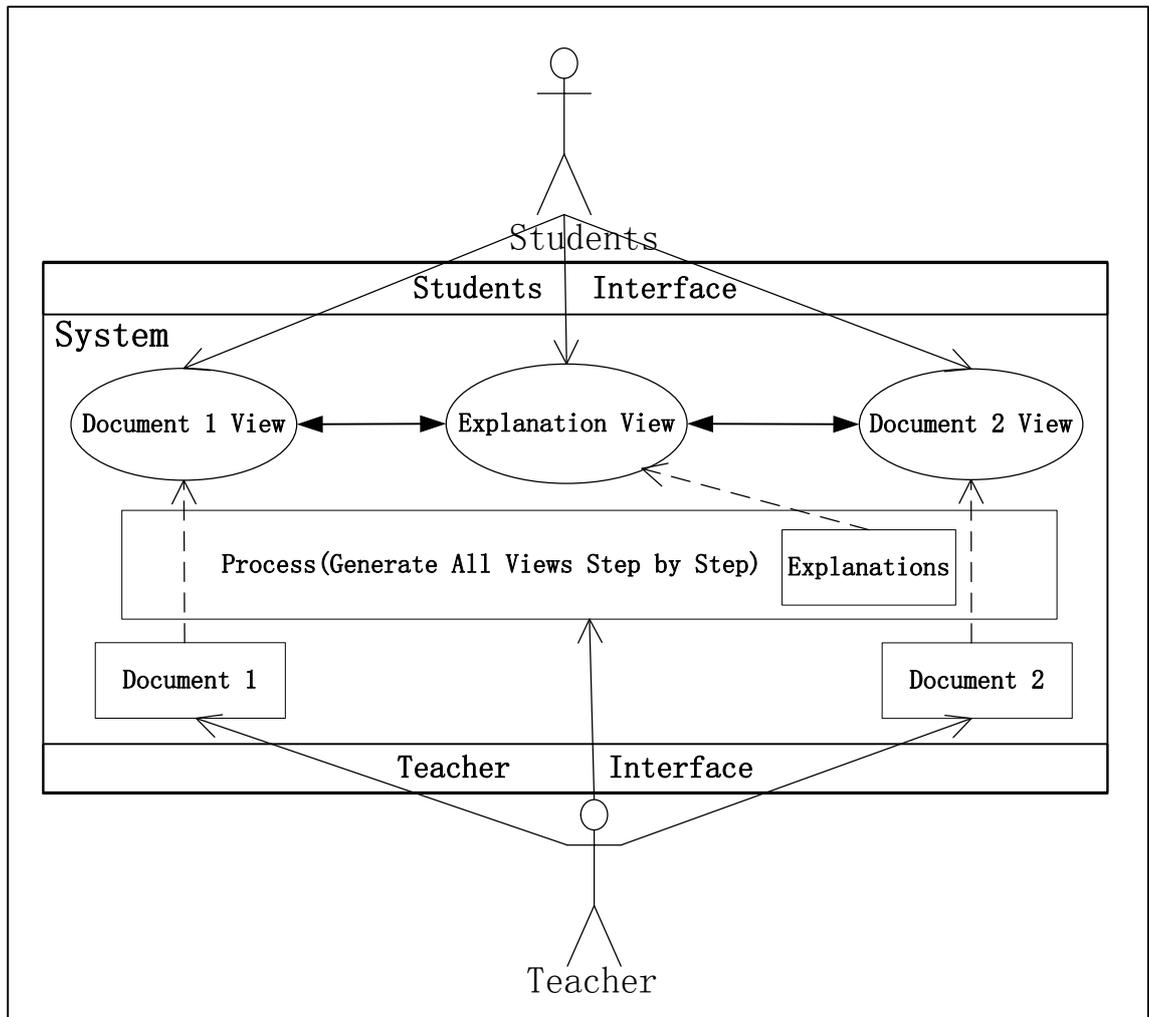


Figure 4.1 Multiple Coordinated Views Structure to Represent Worked Examples Using IWE

For example, a teacher would like to create a worked example to demonstrate how an ER diagram can be generated from the requirement description. The Document 1 View can be used to represent the requirement description document, and the Document 2 View can be used to represent the ER diagram document. Explanations will be edited during the creation of process step by step, and then be shown in the Explanation View. IWE can support more than 2 document views, but to keep the structure diagram in Figure 4.1 simple only 2 document views are shown. For example, if the teacher would like to further demonstrate how related SQL commands can be written to extend the previous example, the SQL commands document can be shown in a Document 3 View.

If faded work examples are being used, then the teacher interface must allow the teacher to insert questions into a process and the student interface must provide a mechanism for the student to answer these questions and store the students' answers. These answers can be provided as feedback to the teacher.

Additionally, a feedback mechanism is required to allow students to send questions or comments asynchronously to the teacher via the system. Collected feedback from the students is returned to the teacher, allowing them to modify interactive worked examples, for example, to correct errors, extend worked examples or improve explanations, if necessary.

4.1.2 Basic Concepts of IWE

Some basic concepts need to be defined before considering detailed requirements. As mentioned before, the worked examples produced by IWE focus on transformation between representations and showing the relationship between representations. Hence, IWE must support the definition of documents that are either graphical or textual, as representations. In order to highlight the relationships between different parts of representations, the documents must be constructed of small pieces, called fragments, which can be graphical or textual. They are the fundamental components of a document. Hence, a document is built from a collection of fragments, for example, a collection of graphical fragments becomes a graphical document and a collection of textual fragments becomes a textual document. A graphical fragment could be a shape or a line with its unique attributes, like size, colour and so on. For example, Figure 4.2 shows three different graphical fragments. Fragment 1 and Fragment 2 are of the same type: they have the same attributes, but with different labels. Fragment 3 is a different type of fragment with different attributes and label compared to fragment 1 and fragment 2.



Figure 4.2 Graphical Fragments for Building Graphical Documents

In order to define a fragment, these attributes need to be encoded. Fragment type is defined to manage the attributes of a fragment. A fragment could be a graphical or textual, so the fragment type is either a graphical fragment type or a textual fragment type. Because a document consists of many fragments which may depend on different fragment types, a document type can be understood to manage a collection of fragment types which are used to describe a document.

In order to show transformations and allow the linkage of contents between representations to be highlighted automatically during the interaction, different documents and individual fragment relationships between documents need to be managed. An application is defined as a combination of different documents within a multiple coordinated views structure. A process is defined as presenting a sequence of related fragments from different documents with associated explanations in order to show a worked example interactively. A document can be used in more than one process within an application. A worked example is one process within an application. There should also be a mechanism that allows related fragments from different documents to be highlighted to show correspondence. Correspondences should be highlighted automatically if an individual fragment of a correspondence is triggered by selecting it. Taking the previous requirement to ER diagram as an example, a correspondence can be between a noun in the requirement document and an entity (labelled rectangle) or an attribute (labelled oval), related to this noun in the ER document.

A student should be able to explore the worked example and interact with the documents in several ways, either by stepping through processes, which consist of a series of steps to display parts of the documents, or by interacting with the fragments of documents and observing relationships to other documents.

4.1.3 The IWE Data Model

Figure 4.3 shows the data model of IWE. It is organized into three layers: the Document Types, Documents and Applications layers. The Document Types layer is the fundamental layer of the whole model; it describes the types of fragment, either graphical or textual. The Documents layer is the core data layer of the whole model; it defines either graphical documents or textual documents, which contain a set of fragments based on the description of fragment types in the document types. The application layer of the model allows documents to be allocated into a predefined multiple-panel structure and stores the representation of processes. A process involves: describing the sequence for presenting different fragments of documents in different panels; defining the relationships between different fragments in different documents; and adding the teacher's explanations for each step in the process.

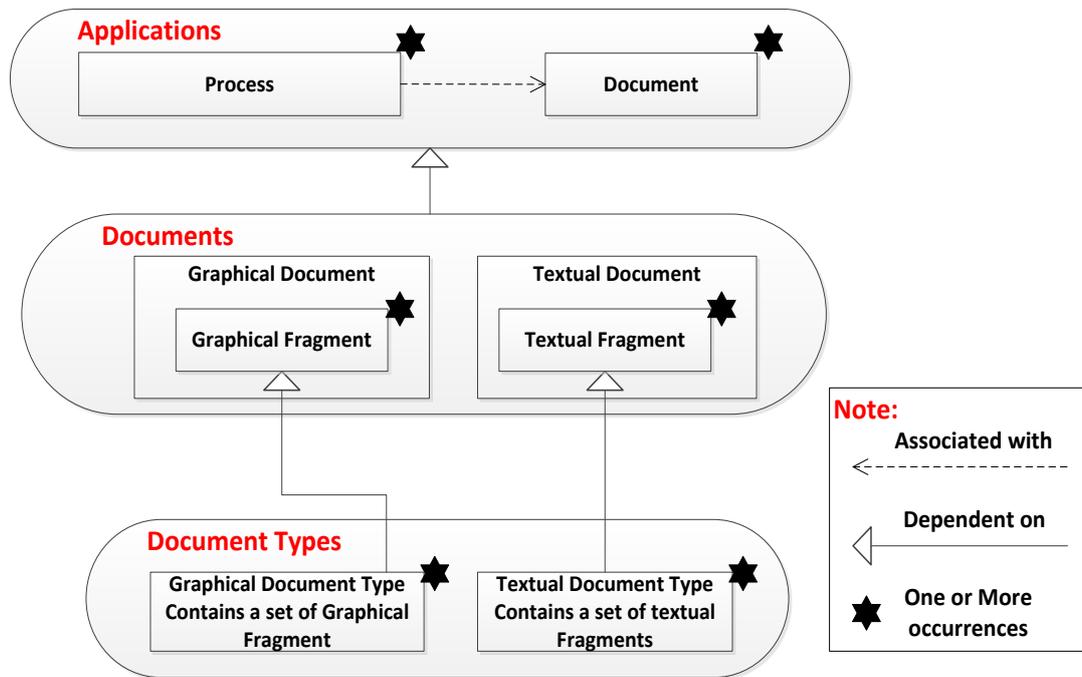


Figure 4.3 Data Model of IWE

Figure 4.3 defines the data model for building a worked example and supports the core function of IWE, presenting interactive worked examples to students. Student details, their feedback, answers to embedded questions and log files are stored in the system. This data is associated with a worked example, but not directly related to how the worked example is built and presented. Storage requirements for this data are defined in 4.4.1 System Requirements and 4.4.2. Log Requirements.

4.2 Detailed Requirements

The detailed requirements are described separately based on each type of user's individual requirements.

The teacher's user interface requirements require the ability to:

- Create fragments for displaying the textual and graphical documents and provide editors for the documents.
- Define a multi-window environment with each window consisting of a panel to hold a document. Allocate documents to panels in this interface.
- Describe the fragments of several documents as requested through a step by step process.

- Create the explanation of each step in a process.
- Preview an interactive worked example as it will be seen by the student.
- Describe how fragments from two representations are equivalent. If two equivalent descriptions exist, correspondences between elements of each can optionally be identified.
- Display student feedback, students' answers to questions and log files of students' usage.

The students' user interface requirements require the ability to:

- Allow the student to select a worked example to explore.
- Display the contents of documents in a display panel that is defined by the teacher.
- Control the step by step representation of documents, including start, forward and back.
- Show explanations while demonstrating the required contents of documents.
- Allow the student to give feedback, either comments or questions, to the teacher.

4.3 Use Case Descriptions

Based on the basic requirements discussed above, a use case diagram shown in Figure 4.4 describes how teachers could build interactive worked examples and how students could access the interactive worked examples through the system. Teachers and students have different purposes for using the system and the functions and the scope of the system which they can use are different; therefore two different user interfaces must be provided. The teacher's interface mainly focuses on building interactive worked examples in an authoring way and the student's interface mainly focuses on supporting exploring and interacting with interactive worked examples. Figure 4.3 give an overview of the main functions (use cases) of IWE and the details of each use case will be explained in the following sections.

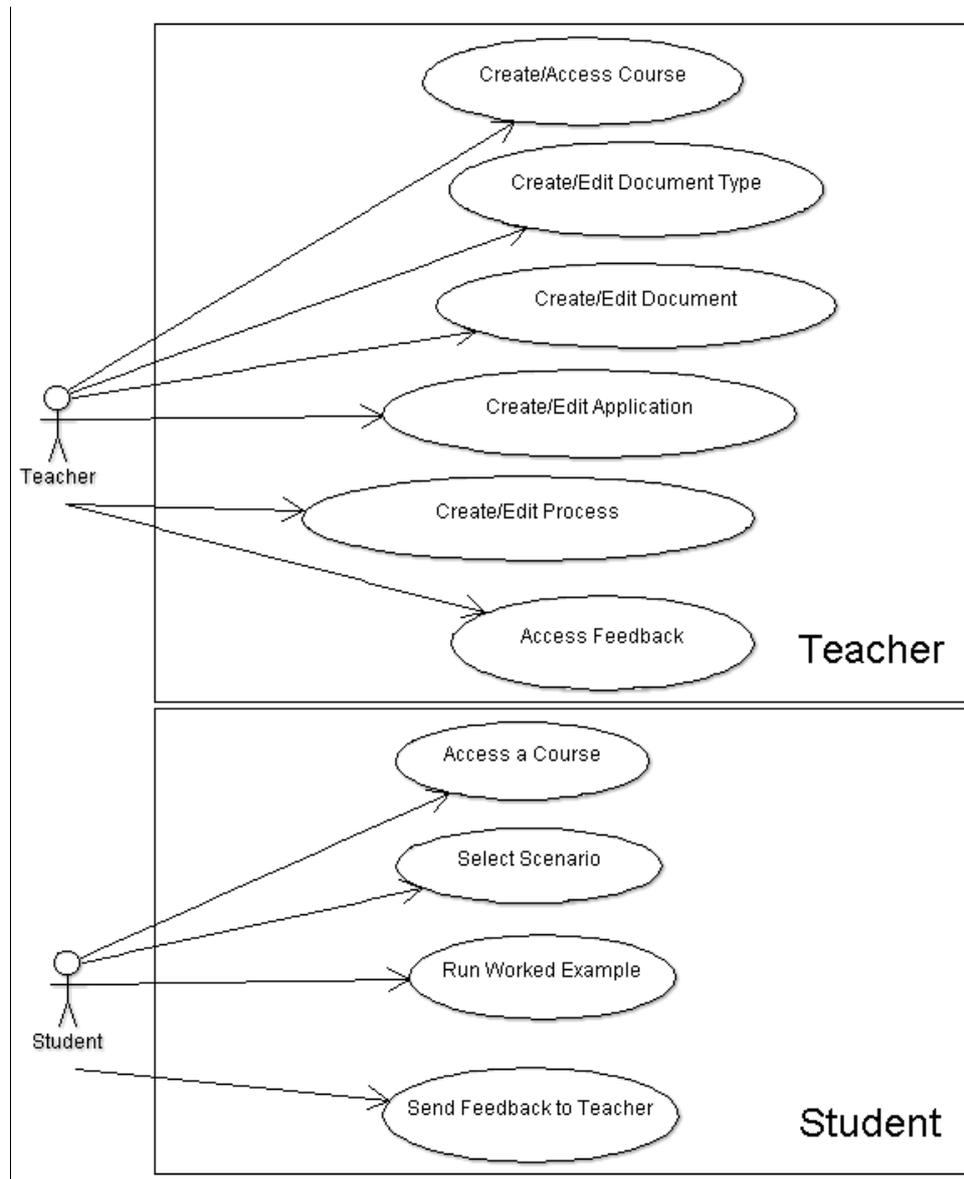


Figure 4.4 Use Case Diagram of IWE

4.3.1 Teacher's Use Case Descriptions

4.3.1.1 Create/Access Course

The first use case is to create/access course. If a teacher is accessing this system for the first time, it should ask the new teacher to register their course name and the system will create a folder called “course name” for the teacher. If a returning teacher would like to create worked examples for a new course, he needs to register this new course name. Hence, a field for entering course name is required in the register user interface. This user interface should allow a returning teacher to select an existing course name to access the system, if they are adding more or modifying worked examples for that course.

4.3.1.2 Create/Edit Document Type

The second use case is to create or edit a document type. It is vital to capture the semantic nature of the documents being manipulated as well as the syntactic aspects. It is also important to capture a meta-model of each document which is the document type. Hence, the first step for a teacher to create a worked example is to define the document type. A document type describes and names the kinds of fragment types that make up the document and how they can be inter-related. It is a collection of fragment types. Creating a document type starts by choosing whether it is textual document type or graphical document type.

Create/Edit Document Type contains two tasks which are create a new document type and edit existing document types. There are two different document types, so it requires two different user interfaces to create graphical document type and textual document type; the defined document type should be able to be modified by using the same user interfaces.

The graphical document type user interface should allow a teacher to define a set of graphical fragment types, which are the node types and arc types. The node type will be some the regular shapes which could be triangle, rectangle, and ellipse and so on. The node type can be filled in different colours or transparent, the border of the node can be set up a different colour, and the name of the node can be set up a different colour as well. The arcs will be different types of lines which will be used to connect nodes, such as single line, double line, arrow line, dotted line and so on. The attributes of these lines can be changed, such as colour, width and so on. The arc may have a beginning point and a finishing point, which are usually points on two different nodes. However, an arc by default should be able to start from any node and finish at any other node. The name of arc could be labelled, if needed.

The textual document type user interface should allow a teacher to define a set of textual fragment types. Each single textual fragment type is a kind of textual style, which allows setting up the colour, font, size, and emphasis (bold, italic, underline) of the text. This operation could be treated as a predefined textual style which will be applied to pieces of text in a document, for example,

Heading 1, Heading 2, Title, Normal and so on. It should also provide a default textual style for teacher, if the teacher does not need a specific style.

Defined fragment types are saved in document type, which should be able to be reused. This indicates that a document type will be used as a template when creating documents later on. Several documents can share a document type; however, a document does not need to use all the fragment types within a document type. For example, a teacher can set up a graphical document type for describing an ER Diagram. This ER Diagram document type can be used to define several different ER diagrams. The teacher could also set up a textual document type for describing a program.

A document type will record these different fragment types, so a save function is required to save the document type data. If editing an existing document type, it should offer the choice either to save as a new document type or to overwrite the existing document type.

4.3.1.3 Create/Edit Document

The third use case is to create or edit a document. The teacher will apply existing document types to create documents. A document is a collection of fragments, which are instances of applying fragment types. Creating a document is a process of adding different fragments. Creating a document starts by choosing whether it is a textual document or a graphical document. If a new graphical document is to be created, it will be built and based on a selected graphical document type. If a new textual document is to be created, it will be built and based on a selected textual document type. Hence, it requires two different user interfaces to build different types of documents. This user interface can also be used to edit an existing document.

The graphical document user interface should contain palettes to list all the node fragment types and arc fragment types separately. It should also contain a drawing panel for constructing the graphical document. By simply selecting one fragment type, an instance of selected fragment type should appear at the selected position in the drawing panel. Different instances of the same fragment type should be able to be given different labels. A teacher should be allowed to

use an instance of an arc to link the nodes through mouse drag and drop. Some joint points should be provided when joining nodes with arcs. It should allow the arcs to be named differently when connecting to different nodes and the label of the arc will be shown in the middle of the arc. After adding the graphical fragments, the graphical document user interface should support modifying nodes and changing the position of any node through mouse drag operations.

The textual document user interface should contain an area for input and formatting of plain texts. This area is used for splitting a sentence or a paragraph into as many fragments as the teacher requires. It should contain another area to sort these pieces of textual fragments into a sequence and to apply a textual fragment type from the selected textual document type to a particular textual fragment. It should contain a preview area to see the results of restructured plain texts. It should provide basic operations (insert, delete, and edit) for the teacher to modify the construction while creating or editing a textual document.

Defined documents should also be able to be reused. This indicates that a document can be shared by a number of different applications. For example, a teacher creates a graphical ER Diagram document. This ER diagram document can be used with a textual requirement description document or with another textual SQL commands document for different teaching purposes.

A document will record all these graphical or textual fragments, so a save function is required to save the document data. If editing an existing document, it should offer the choice either to save as a new document or to overwrite the existing document.

4.3.1.4 Create/Edit Application

The fourth use case is to create or edit an application. The teacher will define a set of documents for the application and may set up the correspondence between selected documents. An application is created by giving a name and choosing a multi-panel layout for displaying documents. The number of the panels could be from 1 to 4. Each panel will be used to display a different

document. The Explanation panel is always included below the display panels for the documents. Hence, a guidance user interface is required which includes:

- First, choose a predefined multi-panel structure, which allows the teacher to decide the structure for displaying documents.
- Second, allocate a document into a panel for display purposes.
- Third, optionally set up a correspondence relationship between fragments in documents in order to highlight these linkages automatically when student interacts with these fragments.

An application will record all these combined documents, the multi-panel structure and correspondences, so a save function is required to save the application data with a unique application name. If editing an application, it should offer the choice either to save as a new application or to overwrite the existing application.

4.3.1.5 Create/Edit Process

The fifth use case is to create or edit processes of an application. Processes are created in the context of an application and consist of controllable changes to the documents in the application. A process must be based on an application. This means one application could have several processes, where each process is a complete worked example. A process can be understood as an interactive display which represents the contents of different documents in a sequence. Hence, a process is made up of a sequence of steps, executed one at a time, and a step is made up of a set of changes all executed at the same time. A change is a fragment from a document with an action. The actions could be inserting, remove, highlight or unhighlight a fragment, show the whole document or ask a question. Because several changes can happen in one step, an explanation of each step is required. This should help students to understand why these changes happen together. The purpose of a process is to generate a worked example based on the documents in the application step by step with explanations.

Hence, creating a process is to setup an interactive demonstration of different contents from different documents with teacher's explanations. It requires a user interface to manage the sequence and the contents of demonstration and

to add explanations during the representation. Because an application could contain several processes, before creating a process it should ask to which application the new process belongs and give a name to the new process.

The process construction user interface should allow teacher to define steps, changes and explanations. It should contain an area to show the structure of the process. It should allow teacher to select any fragments from any document with an operation as a change. Then add a change into a step. It should contain another area, which allows teacher to add an explanation to a step by simple text inputting. It should also provide a preview function, which allows the teacher to see the current working results. Preview results should be as same as student will see when exploring this process.

A process is used to show the document contents within an application step by step, with explanations, and an application could contain several processes. Hence, a single process could only show part of the document contents as required. For example, an application contains two documents, which are an ER diagram document and a SQL commands document. The purpose of this application is to demonstrate how to represent the different relationships between entities in an ER diagram in SQL commands. One process could be used to demonstrate 1-to-1 relationships. This process would only use part of the ER diagram document, which shows two entities and the 1-to-1 relationship between them and use part of the SQL commands document, which shows relevant SQL commands. Another process could be used to demonstrate 1-to-N relationships; this will use different parts of the ER diagram document.

A process will record all these steps, changes and explanations, so a save function is required to save the process data. If editing an existing process, it should offer the choice to save as a new process or to overwrite the existing process.

4.3.1.6 Access Feedback

The sixth use case is to access feedback consisting of: students' feedback sent while using the tool; students' answers to questions in faded worked examples; and students' usage log files. This could help teachers to modify the worked

examples and improve the worked example design. A user interface which can sort and display all students' feedback is required together with a mechanism for summarizing students' answers. This user interface should allow the data to be sorted by individual student or by worked example. For textual answers, the data can only be summarized by presenting the related answers grouped together for the teacher to analyse. For multi-choice questions, the data can be summarized to show which questions were answered correctly and the wrong choices for those answered incorrectly. A user interface for accessing usage log files is also required.

4.3.2 Student's Use Case Descriptions

As shown above in Figure 4.1, students will access IWE through a student interface. In order to make the student user interface easier to understand the terms scenario and worked example are used in place of application and process, respectively. Using this interface, students should be able to: select a scenario (defined by an application) to explore; pick a specific worked example (defined by a process) to explore; write questions and comments about the worked examples and send these back to the teacher. Hence, it requires a student interface to show scenarios designed by the teacher. Using this student interface, the worked example of the chosen scenario can be explored as the teacher specified. This user interface should contain an area, which allows a student to select scenarios and worked examples. It should also contain another area, which could be used to display the documents of the selected worked example including explanations with the designed structure. It should provide a mechanism to allow students to provide feedback and their answers to any embedded questions.

4.3.2.1 Access a Course

The first use case is to access a course. The user interface should allow students to access a course by choosing the course name. To meet University Ethics Requirements, the user interface should also ask whether students would like to take part in the research project and allow some data about their IWE usage to be recorded. If the student agrees to join the research project and uses IWE in the lab, IWE should record the user name as the student login to the lab

machine. If the student agrees to join the research project and downloads IWE to his own machine, IWE will not be able to record his user name, used in the lab or usage of log data.

4.3.2.2 Select Scenario

The second use case is to select a scenario, which will display a list of worked examples that belong to this scenario.

4.3.2.3 Run Worked Example

The third use case is to choose a worked example and execute it. The documents of the worked example will be displayed step by step with teacher's explanations. The worked example will be executed with the document presentation structure of the scenario plus an extra panel to display the explanation of current step. The exploration process can be manually controlled by the students, for example, go back a step, go forward a step, go back to the beginning or answer embedded questions. A student's answer should be shown in the step in which the model answer provided, so students can compare their answers with the model answers.

4.3.2.4 Send Feedback to Teacher

This is an optional use case, which allows students to ask questions or to comment on the worked examples with a fresh mind, while exploring a worked example. The student may send his personal feedback to the teacher, if he still cannot understand the delivered worked example during or after exploration. When IWE is used in a university laboratory the student's user name can automatically be added to the feedback. If IWE is used outside the university lab, it should ask students to provide their lab user name. This information is used to identify who sent these feedback messages, so the teacher can recognize who sent the problems and reply, if necessary. By asking for the lab user name, feedback and answers of the student can be identified and merged with other data for this student.

4.4 Other Requirements

4.4.1 System Requirements

The IWE should recognize where it is used, either on a lab machine or on the student's own machine.

After students access IWE, they will want to select a course and worked example to explore, so any data required must be loaded by the system automatically.

Teachers must be able to access students' feedback. The communication between the two types of users could be achieved by using email or via a course website to send and receive feedback messages. However, it would be much better to use an always-online database server to take over the sending and receiving of feedback messages and the recording of answers to embedded questions. Such a server could also be used to make applications available to students.

If students use IWE in the lab, their feedback data will be saved as XML files on the server, so the system can load the students' feedback data automatically. IWE should also save student's answers to the embedded questions of faded worked examples into XML files as well. These answers are also important feedback data, which can be analysed for use by the teacher. The information saved into an answer XML file includes: student's user name, the name of the worked example, the type of question, the date of answering, the time cost of answering a question and the answer to a question.

If students use IWE outside the lab, their feedback data and answers to the embedded questions will be sent as attached files by email. After the teacher manually merges these attached files into relevant folders on the server, the system can load all these individual files. IWE should merge all the feedback data based on the course name, student's user name and the name of the worked example.

4.4.2 Log Requirements

IWE should include a log to automatically record how students interact with the worked examples. The original purpose of this logging function was to help evaluation of IWE. However, this usage data also provides feedback to the teacher. Logging should include a timer to record how long a student spends on exploring and interacting with different parts of a worked example. For the process function, it should record every button action, for example, use of next step and previous step buttons and how long is spent on each step. If a student spends a long time on a step this may indicate to the teacher that they are having difficulty understanding the content of this step.

4.4.3 Help Function

A general help function for supporting users to use this system should also be provided for users working independently and looking for support while using the tool.

Chapter 5 Design and Implementation

Detailed requirements have been presented in the previous chapter, so it is required to design and implement a system to meet these requirements. This chapter discusses the design decisions made to meet these requirements and presents the final implementations of these designs.

Section 5.1 justifies the implementation decisions and explains the deployment architecture of IWE, when used inside and outside the laboratory environment. Section 5.2 describes the initial designs of the major user interfaces for the teacher and the student to access the system to do the actions, which are identified in the Use Case diagram in Figure 4.4. It also proposes the teacher's user interfaces to build the required data for the IWE data model. Some of the initial designs of user interfaces were not good enough, problems were found based on the results of pilot user tests and evaluations, which are reported separately in Chapter 6. These evaluation results were used to make important modifications to the design and these are reported with the aim of demonstrating the evolving process of design. Section 5.3 illustrates the final implementation results of the user interfaces, following the sequence of using IWE, with comments on changes that were made to the initial designs. Section 5.4 gives conclusions to this chapter.

5.1 The Implementation and Deployment Architecture of IWE

5.1.1 The Implementation of IWE

IWE was developed by using JAVA Standard Edition (J2SE). The data resources: worked examples, student feedback and answers to the embedded questions were stored in XML files. The open source Log for Java (Log4J)[106] was selected to record users' activity while interacting with the user interface. Each log file was stored in two formats: textual log file and html formatted file. The textual log file can be used for further analysis by the researcher, and the html formatted file can be review by using log viewer in the teacher interface. Figure 5.1 show how these data resources are used by IWE. Worked examples resources are created by using the teacher interface, however, when students use the

student interface to access IWE application, they are loaded in automatically by the IWE application. The feedback, answers to the embedded questions and log files are created while students explore these worked examples. All these data resources can be accessed by using the teacher interface.

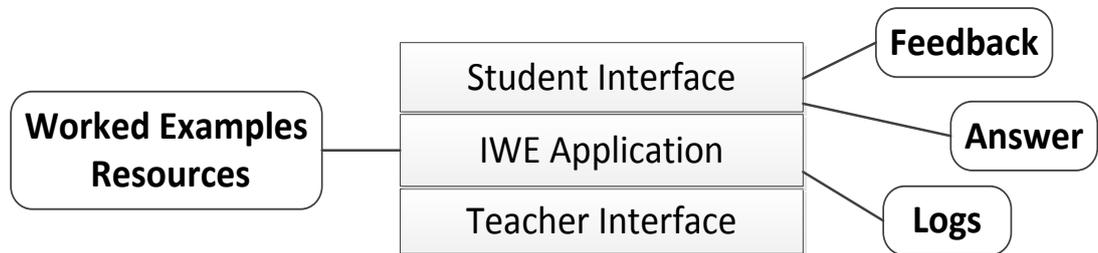


Figure 5.1 Data Resources Used by IWE

Two different user interfaces were implemented, based on the discussion in Chapter 4.3. The detailed UML class diagram of IWE application, which maps the three-layer data model of IWE shown in Figure 4.3, can be found in Appendix 11. The detailed reason for using Extensible Markup Language (XML) technology and the designed XML schema to save data can also be found in Appendix 12.

5.1.2 The Deployment of IWE

Due to student and teacher users needing to access the worked example resource, a server for keeping this set of data is required. This server can also be used to save 3 other sets of data: students' feedback, answers and the log files. The teacher user must be able to access these sets of data, so IWE was distributed by using the traditional client-server architecture. Figure 5.2 shows how the IWE application was deployed for students' users in the laboratory environment.

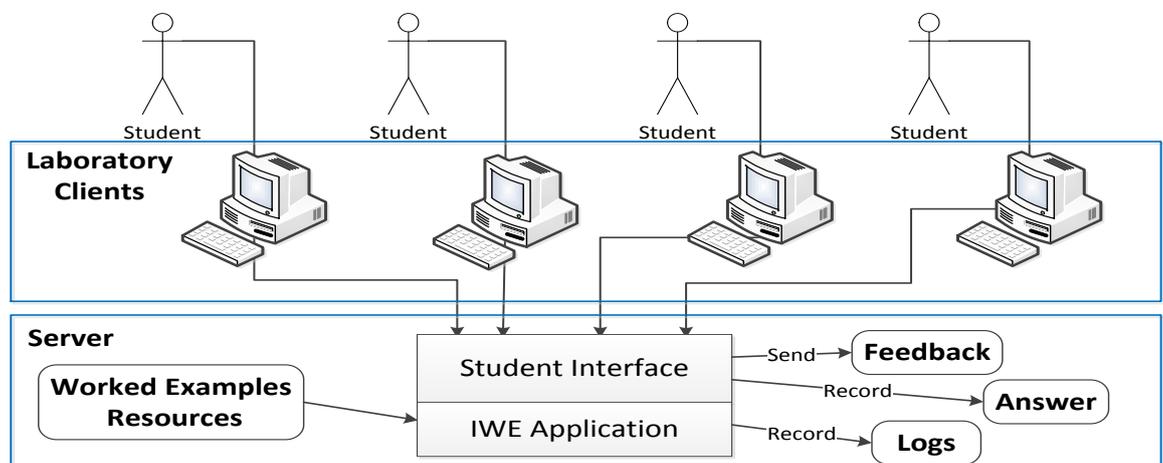


Figure 5.2 IWE Deployment Architecture in the Laboratory Environment

Students access IWE application using the student interface through the laboratory clients. When they open the student interface, the IWE application automatically loads the worked examples resources in. As students explore worked examples, they may answer the embedded questions or send feedback by using the student interface, and these data resources are saved on the server as well. The IWE application can record how the user operates the student interface by recording answers to questions and log files. This operational data is also saved on the server.

A teacher can view feedbacks, answers and log files by running IWE on the server and accessing the teacher interface through their own machine as a client. The teacher can create or modify the content of worked examples by using the teacher interface. Student and teacher users share the same data set of worked examples resources; so if a teacher user creates a new worked example or modifies an existing worked example, student users can receive these updates without deliberately downloading. The student feedback, answer files and log files are saved in predefined folders on the server. The teacher user can use different viewers provided by the teacher interface to see the student feedback, answer and log files, which were stored on the server. Teachers can also build worked examples on their own machines, and then transfer the worked examples resources onto relevant folders on the server. Figure 5.3 shows how a teacher can build worked examples for students to use in the Laboratory Environment. It also shows how a teacher can collect feedback, answers and log files.

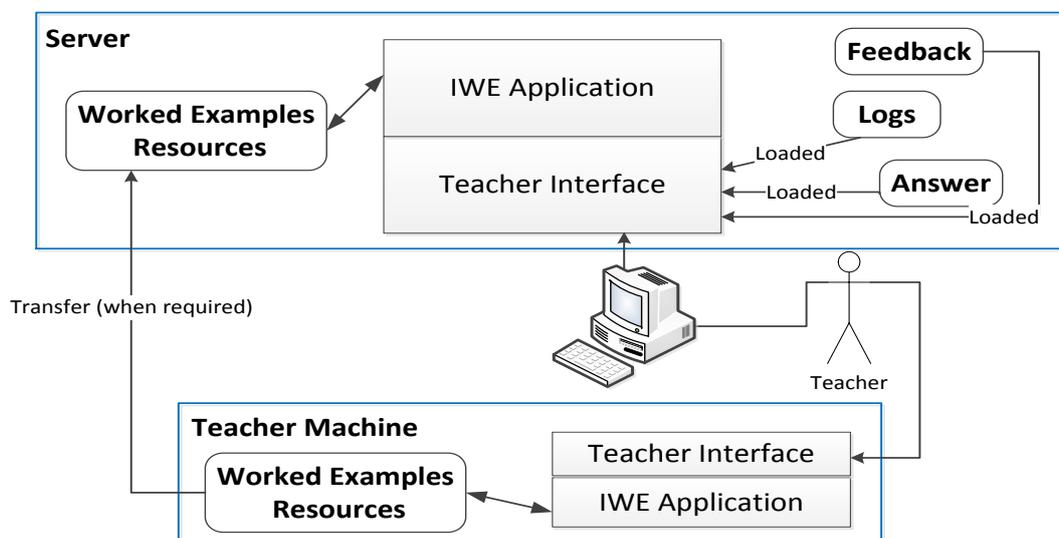


Figure 5.3 Building Worked Examples for Students Using in the Laboratory Environment

IWE can also be used outside the laboratory. The distributed architecture is shown in Figure 5.4. Students need to download the student version of IWE application and the worked examples resources into their computers firstly. Then they can use IWE application in the same way as they used it in the laboratory. However, there is a difference while they are sending feedback or answering questions, which is asking the student user to provide their laboratory login user name. Then the IWE application will automatically use a default Gmail account to send an email without disturbing the user. The input user name is used as title, and their feedback and answer files sent as attachments.

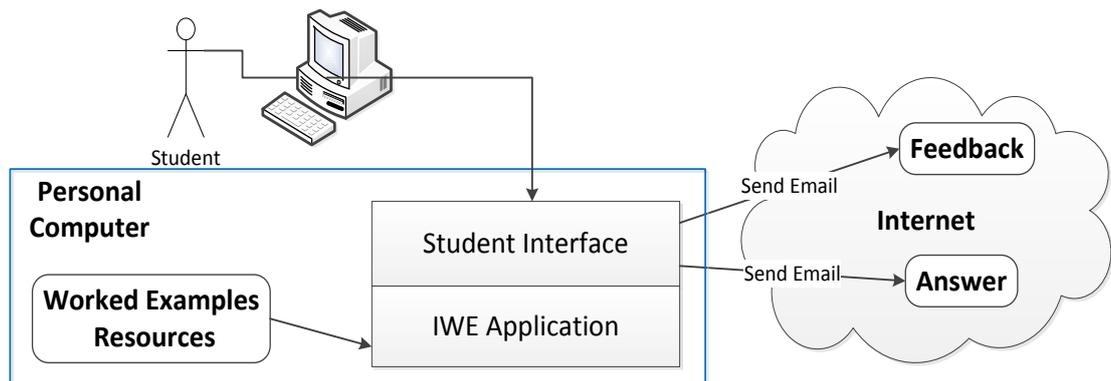


Figure 5.4 Students Use IWE outside the Laboratory Environment

Figure 5.5 shows how the teacher can collect students' feedback and answers if students use IWE outside the Laboratory Environment. The teacher can log in to the default Email account and download these email attachments; then manually merge attachments, based on the titles, into relevant folders on the server. Because teacher and student users do not share the same worked examples resources, if the teacher updates worked examples on the server, the students need to download the new worked examples resources, and then manually merge them into the relevant folder on their own computers.

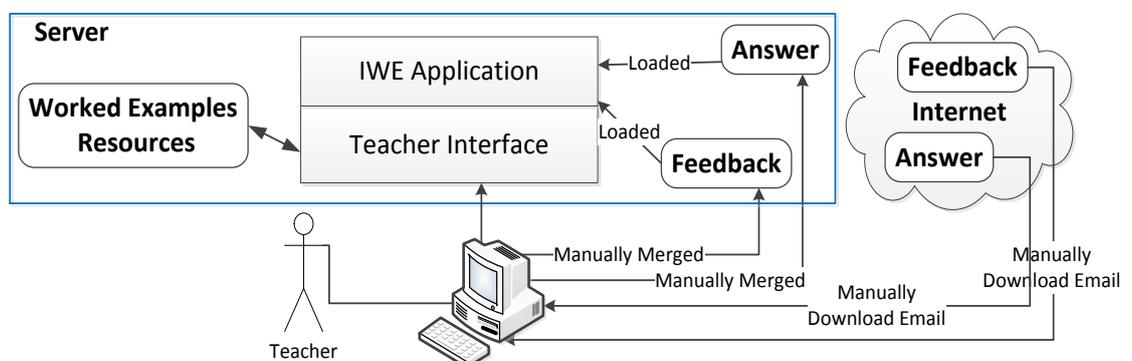


Figure 5.5 Teacher Collects Data when IWE used outside the Laboratory Environment

5.2 Initial User Interface Designs

The use case diagram described in the requirement chapter shows that two different types of users, teachers and students, will use the IWE. Hence, two different user interfaces need to be designed. This section discusses the initial design decisions taken when building these two different user interfaces.

5.2.1 Teacher User Interface Design

IWE is designed to be an authoring environment for teacher to create interactive worked examples through describing documents, setting up the relationship between documents and the fragments, and then representing different documents together with explanations in an interactive way. The teacher user interfaces should allow teachers to: define document types and author documents; allocate documents into a predefined multiple-panels structure to create an application; set up relationship between fragments in different allocated documents and define the sequence of representing fragments of documents, together with explanations. Hence, the decision is made to use a descriptive graphical user interfaces to collect data, which can describe the required documents. By using these types of user interfaces, the teacher only needs to fill in or pick up the required information to finish the creation work.

Figure 5.6 shows a draft design for creating the graphical document type user interface. In requirement chapter 4.3.1.2, it explained two kinds of graphical fragment types need to be created, which are node type and arc type. The node type requires data to describe the attributes of a node, for example, a rectangle should be defined with its height, width, border colour, filled colour and so on. The arc type requires data to describe its starting and finishing position, the colour, and the width and so on. Hence, the creation of node fragment type and line fragment type should be separate. Some popular node types, like rectangle, diamond, ellipse, should be provided in a shape list; and line types, like plain line, dotted line and arrow line, should be provided in a line type list. This kind of selection limits the user's choice. However, if a new shape or a type of line is needed for a particular user, these two lists can be extended easily as requested. This selection method for defining a shape or a line is a little complex compared with the dragging and dropping method from a

comprehensive shape template, used in current commercial design tools, like Microsoft Visio or Smart Draw. This initial design is only a prototype, and aims to evaluate research questions. It can be modified and updated in the future, if necessary.

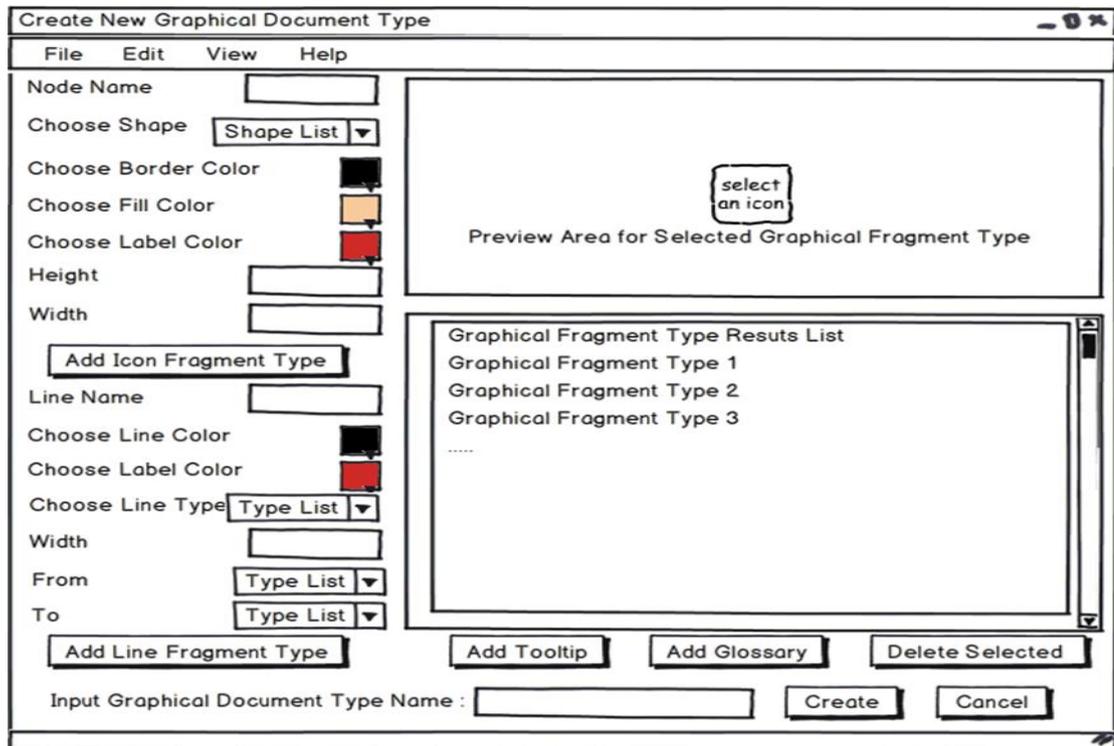


Figure 5.6 Graphical Document Type User Interface Design Draft

To define a target shape, requires the user to fill in some attribute information to describe the shape, such as colour, size. During this process, the user only can imagine the defined shape, so the user interface should allow previewing the created results. Therefore, the user interface needs a list to keep the created results and an area to preview the created results. For example, selecting a fragment type from the results list, the defined result should be shown in the preview area, which is shown in the right top area of Figure 5.6.

Defining a textual document type is a quite similar procedure using another similar user interface, so it is not discussed. However, the textual document type editor user interface was modified slightly, based on two teacher users' suggestions, for example, allowing previewing the defined textual style result, and allowing the existing textual style to be updated rather than deleting it and then creating a new one. The draft design of textual document type editor user interface is shown in Figure 5.7.

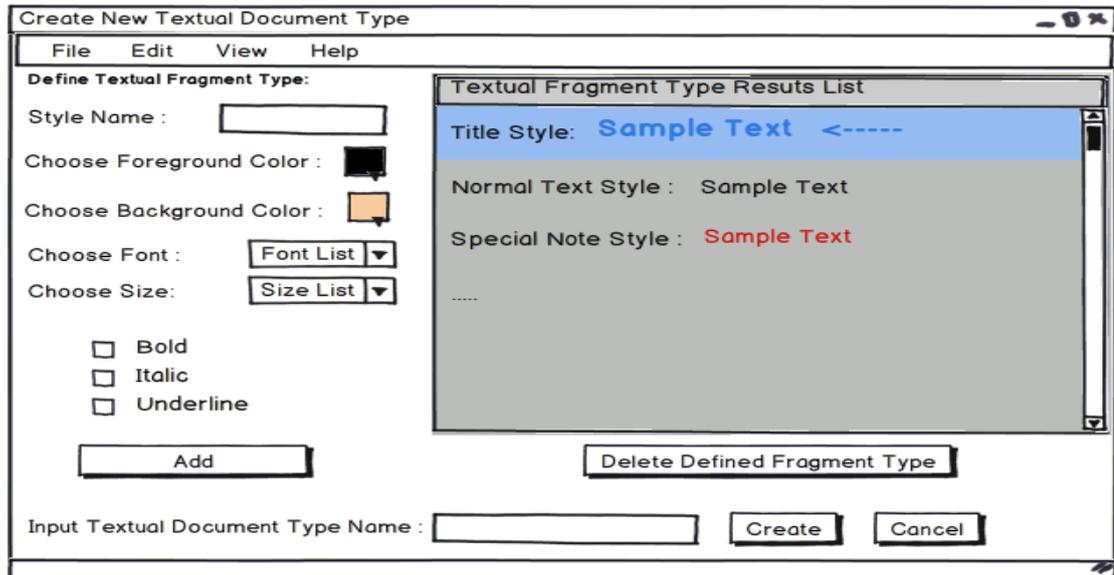


Figure 5.7 Textual Document Type Editor User Interface Design Draft

The next step is to use these defined document fragment types to define a target document (as described in section 4.3.1.3). A document is a collection of fragments. Using textual document type to create a textual document as an example, each textual fragment type can be applied to several different textual fragments. These textual fragment types are like simplified forms of the styles in Microsoft Word. The text document when displayed looks quite similar to a Word document; however, the structure of these two documents is different.

The textual fragment is the basic element of a textual document, which could be a single word, a space, several words, a sentence or even a new line symbol. For example, a paragraph in a Word document contains 30 single words, if displaying the same document as a textual document defined by IWE, it may be only a single textual fragment, 10 textual fragments, 30 textual fragments or 50 textual fragments. One paragraph in a Word document can have different styles, which depend on the selection of starting and stopping positions. Textual documents defined by IWE can be formatted with different styles by using different textual fragment types for the textual fragments. Following this idea, an initial textual document editor was designed, which is shown in the Figure 5.8.

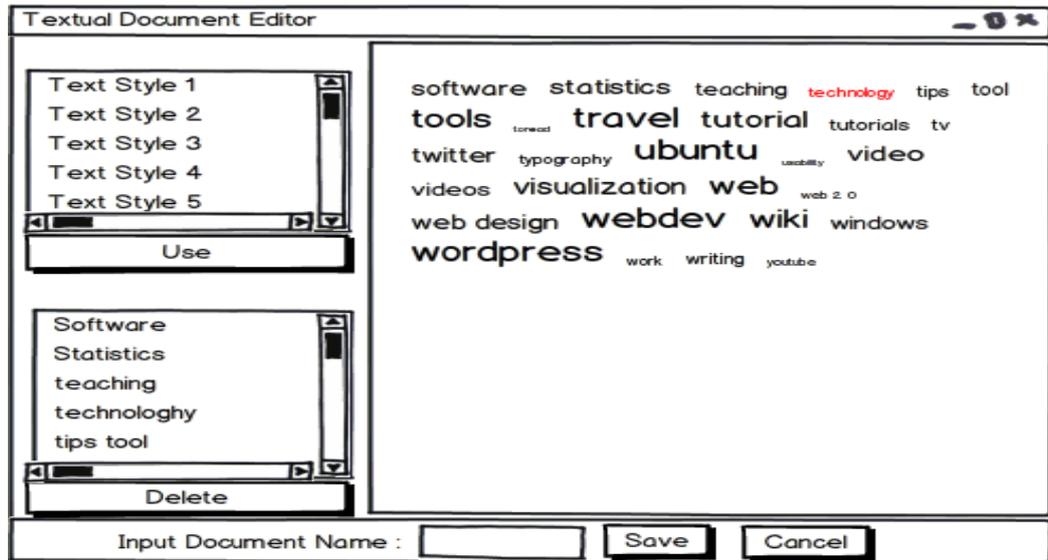


Figure 5.8 Textual Document Editor User Interface Design Draft 1st Version

However, based on a pilot teacher test, this version of the design was not efficient enough for inputting text contents and not convenient enough for editing and formatting. Hence a new version of the textual document editor was designed in order to overcome these identified drawbacks, which is shown in the Figure 5.9.

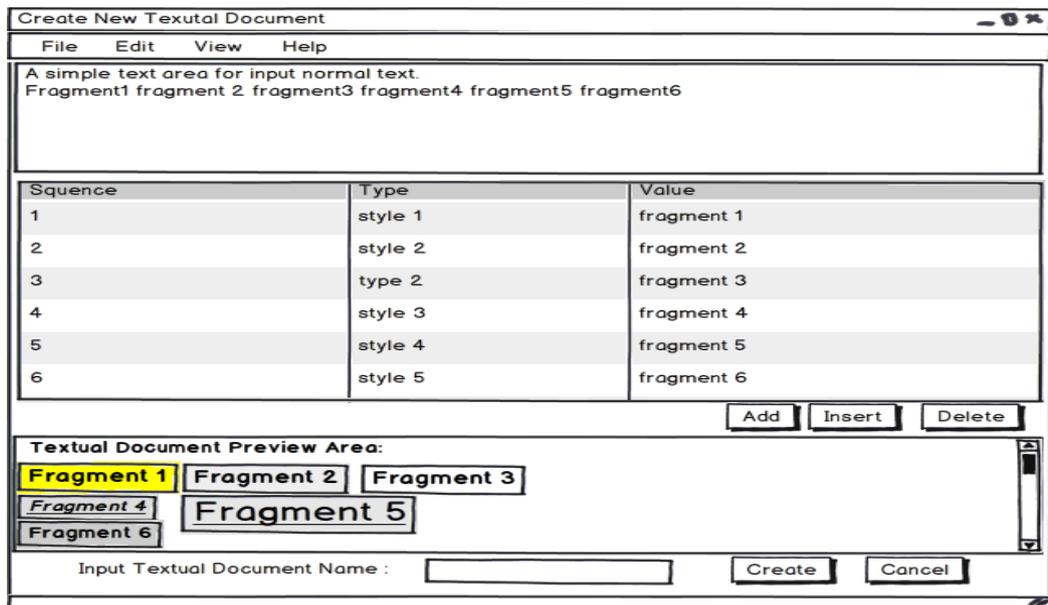


Figure 5.9 Textual Document User Interface Final Design Draft

This new design contains three parts from top to bottom, which are text input area, formatting area and preview area. The text input area is similar to a normal text editor, it should capture all the text input from the keyboard or pasted as plaintext from another digital document. This feature allows user to reuse their existing worked examples contents, which could speed up the input of text. The next task is to split this source text into pieces as textual

fragments. Hence, the user interface must capture the user selecting starting point and stopping point, and then record this selection as a new textual fragment. By clicking the “Add” button, this selected piece of text should become a new textual fragment and should be shown in the table list in the formatting area and preview area with its default style. If a new style needs to be applied to this textual fragment, a new predefined style can be chosen from the type list in the formatting area table list. The result of applying this new fragment type will be shown in the preview area. The user interface should also support a faster splitting of text; if very few fragment types need to be applied. It should recognize every single line of the source text in the input area as an individual textual fragment. Insert and delete textual fragments should be also supported during the creation procedure. This splitting of contents into textual fragments and then formatting each individual textual fragment was much more efficient than the previous design.

A graphical document editor was also designed, which is used to describe a graphical document. The user interface is quite close to the common drawing tools; it lists the available shapes and lines on the left side of the frame and contains a drawing canvas on the right side of the frame. The user can select a shape or a line first, and then add it into a specific position in the canvas. The design of this interface does not have any special aspects that need to be explained, so it is not discussed.

Creating an application is a process, which combines documents into a predefined presentation structure (see section 4.3.1.4). Hence the application creation user interface is designed as a wizard, which contains two steps. It will ask the user to select a presentation structure first, which can be two panels, three panels or even four panels, and then ask the user to allocate documents into panels; finally the summary of the allocation will be shown to the user, which means the user should be able to preview the allocated documents.

Correspondences can be defined at this stage, through selecting related fragments from different documents. The design of selecting predefined presentation structure, allocating documents into the structure user interface and defining correspondences do not have any special aspects that need to be explained, so they are not discussed.

The process is the key point for creating worked examples within an application (see section 4.3.1.5). It is used more often than other components of the teacher user interface. Figure 5.10 shows the first draft design of the process editor. Defining a process should allow the user to manage the sequence of displaying parts of the example and to edit the contents of a step. A tree structure is chosen to manage the sequence of displaying, which is shown in the left side of Figure 5.10. This structure clearly lists the relationship between steps and changes. Tree nodes (step or change) can be inserted or deleted by using relevant buttons. The tree structure also can be automatically refreshed if any modification is made.

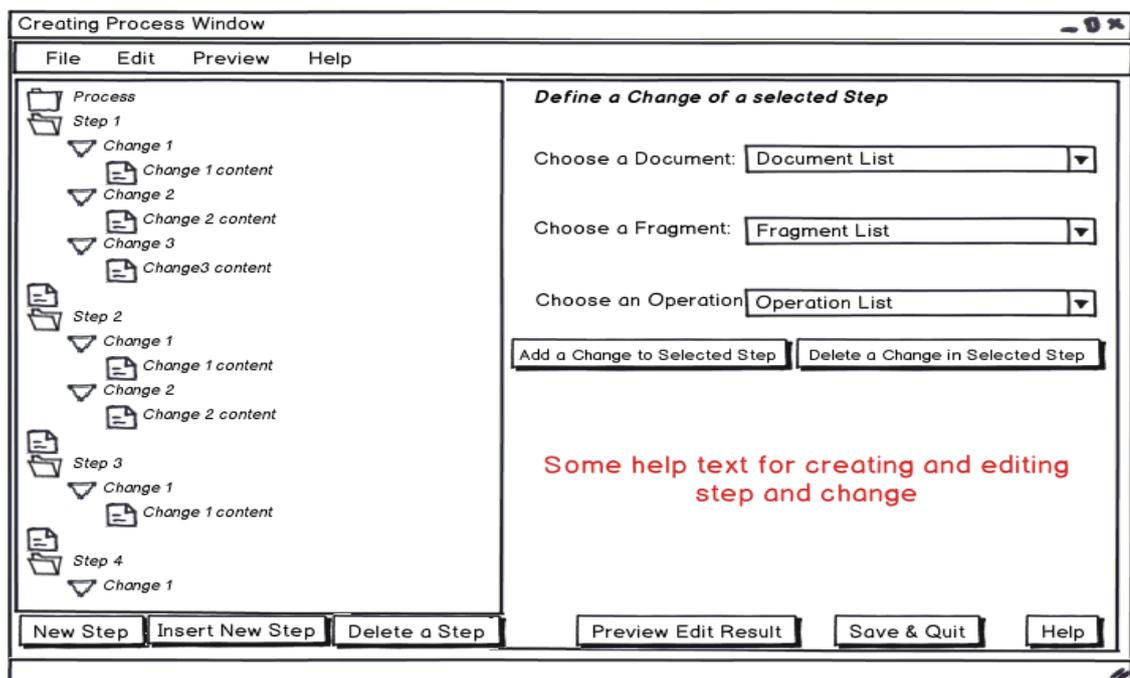


Figure 5.10 Process Editor User Interface Design Draft 1st Version

The change is the key point of creating a step, as it relates to presenting the contents. A change is a fragment of a document plus an action, so the user interface should allow selection of a fragment from a document and adding an action to the selected fragment. An action is an operation on a fragment, which could be: add, delete, highlight, ask question and so on. Hence, three combo boxes are designed to create a change for a step. First, the user decides to choose a document from the document list under the Choose a Document combo box, and then all the fragments which construct this document are sorted in sequence in the fragment list under the Choose a Fragment combo box. Second, the user chooses a fragment. Third, the user chooses an action from the operation list under Choose an Operation combo box to finish defining a change.

Finally, select a step in the tree structure and then add this change into the selected step. Because a step can contain several changes, continuing to add changes just needs the user to repeat these selections. Explanations need to be added to each step, so the user interface should provide an area to do this.

All these operations result in a process that should be shown in a process tree structure. The first level process tree node is a step, the second level process tree node consists of changes and an explanation and the whole tree is a completed process. This process will be shown as an interactive sequence of steps with explanations to students. In order to check the created results, a preview process should be provided. Preview process should be displayed exactly the same as to students. This could help to improve the quality of the process.

Based on the pilot teacher test, two big usability problems were identified, which were related to creating explanations for a step. First, it was required to create a separate textual document to keep all the explanations for each individual step. It was quite hard for the teacher to prepare this separate textual document before creating a process. Second, in most cases, the explanation for a step is unique. If creating several processes, the teacher needs to prepare all the explanations for different processes in one explanation document. It was even harder to prepare this explanation document. Hence, two adjustments were made, which were

- There was no need to create a separate textual document for explanation purpose, the explanation text should be a part of a step;
- Explanation text should be added in while defining a step of a process.

Hence, an improved process editor was designed to overcome these two problems, which is shown in Figure 5.11. Explanation text becomes a sub node of a step node and each step contains a unique explanation, which is shown in the left side of the editor. And add an area for the teacher to write the specific explanation for a step, which is shown in the right bottom side of the process editor. In this new design editor, similar function buttons are integrated, for example, “Delete a step” and “Deleted a Change in Selected Step” buttons become a “Delete Selection” button; “New Step” and “Insert New Step” buttons become an “Add or Insert New Step” button.

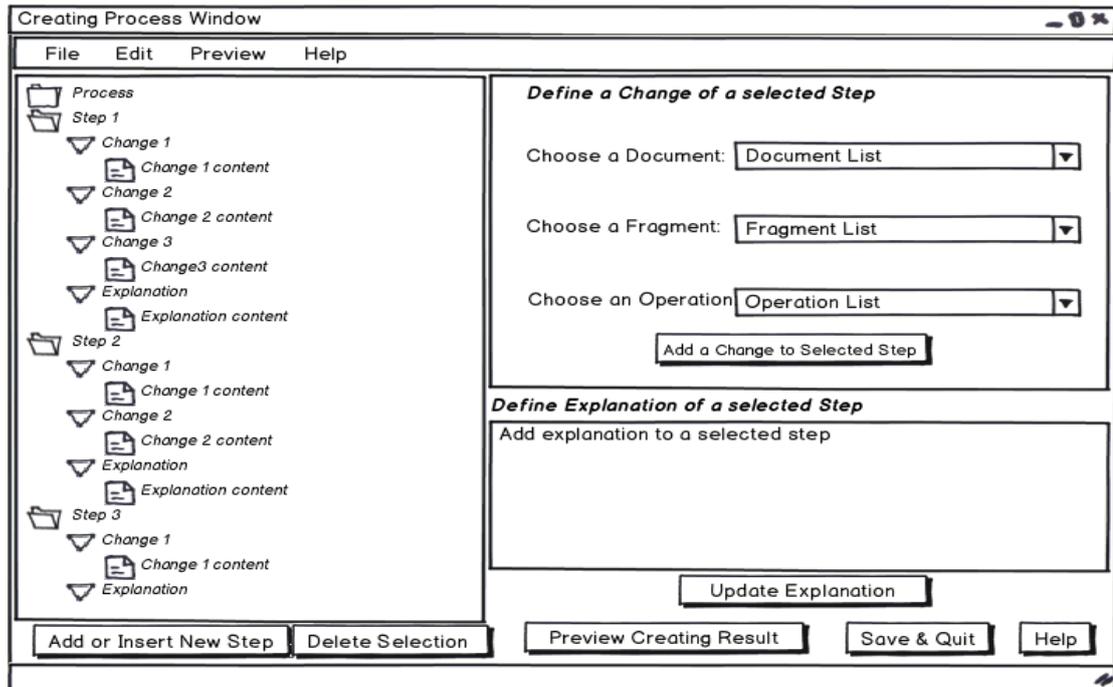


Figure 5.11 Process Editor User Interface Final Design Draft

Up to now, the designs of all user interfaces for creating a basic interactive worked example in IWE have been discussed. Every single user interface is used to finish a part of the creation job of the data model of IWE. IWE is an authoring environment, which organizes these parts together, so each single part can be understood as a function. For a teacher, it should provide a main entry to access all these different functions.

Because the IWE is a development environment that was developed using Java in the Eclipse IDE, which is another popular development environment shown in Figure 5.12, the design of teacher's main entry user interface borrows the idea of the Eclipse IDE interface design. The main window of Eclipse can be arranged in advanced layouts using split windows, tab windows and floating windows in order to manage hierarchical file structure and access editing of each hierarchy. This well-tested design is suitable for working with multiple documents. The main entry of IWE should allow accessing different functions easily.

Another reason for designing the teacher main entry user interface like the Eclipse main entry is to use existing work. InfoNode [107] is a pure Java Swing based docking windows framework. By using this framework, it allows a powerful user interface to be built without so much coding effort. InfoNode also offers a free license for non-commercial development.

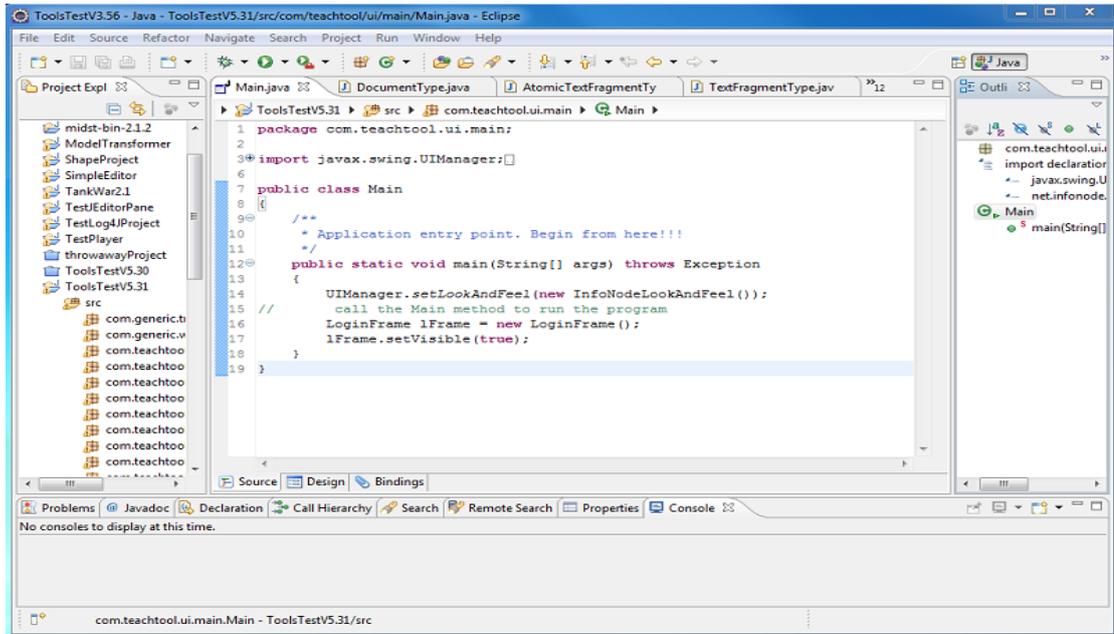


Figure 5.12 Screen Shot of Eclipse Development Environment

Hence, a draft design of main entry of IWE for teacher was constructed which is shown in Figure 5.13. The left side window named Application View is used to list all the applications, with their documents and processes; the right side windows, named Document Type View and Document View, are used to list all the resources files; and the middle part of the window is used to edit or view these resources. For example, if teacher selects Doc 1, which is a textual document, and would like to view the contents of Doc 1, the contents of Doc 1 should be displayed in the centre of the window; or if he wants to edit it, the user interface shown in Figure 5.10 should be displayed in the centre of the window. It also includes a shortcut tool bar on the top of the window to speed up the access to different functions.

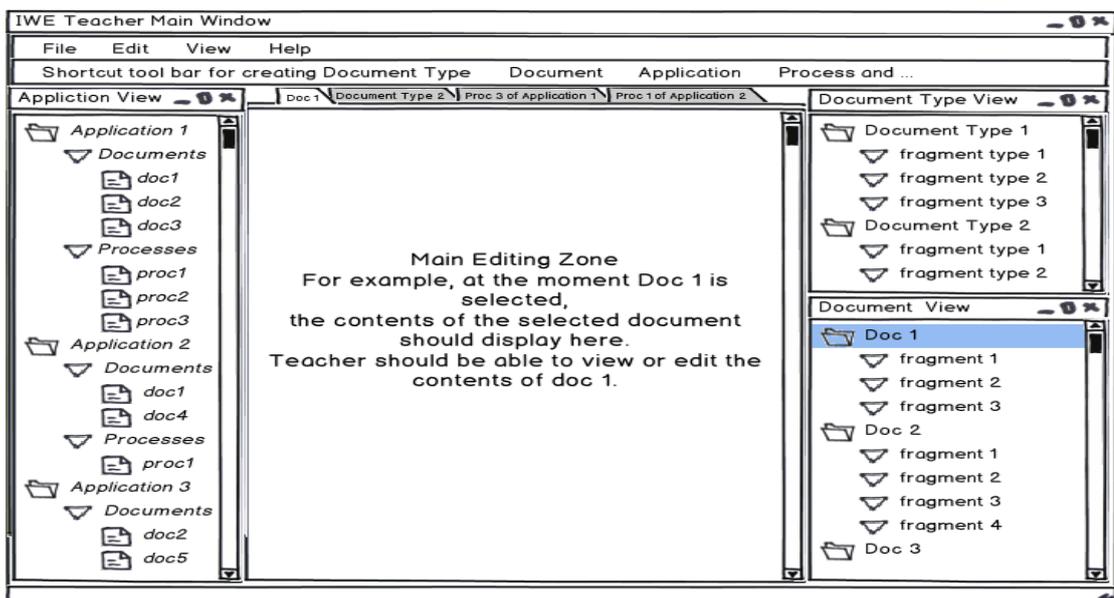


Figure 5.13 Teacher User Interface- Main Entry Draft Design

5.2.2 Student User Interface Design

The use case diagram (Figure 4.4) shows that there are tasks for students to access and explore worked examples and send feedback about them. Students do not need to know the structure of the application, only the teacher needs to be aware of the documents and document types required to build processes. Hence, this information should be hidden from students. For students, they only need to see the name of a *scenario* (application) and the names of the worked examples (processes) within that *scenario*.

A main entry user interface is designed for students to access all the worked examples. Figure 5.14 shows the draft design of the user interface. This design of main entry is similar to the teacher's main entry; so some development work can be reused. For example, the left side of the window, named Scenario View, lists all the scenarios and in each scenario lists all related worked examples. However, it does not have the shortcut tool bar compared with teacher's main entry, because students do not need to create or modify the worked examples. The centre of the window is used to view the selected worked example. However, it will show some welcome and introduction message at first and will be replaced by the process player if a worked example is selected. Because while viewing a worked example, a step may ask student to answer a question, the student's answers will be recorded in the right side top the window, named Answer View. The right side bottom view, named Feedback View, is used to record their problems during the self-learning process. Later on these problems can be sent back to the teacher as feedback messages.

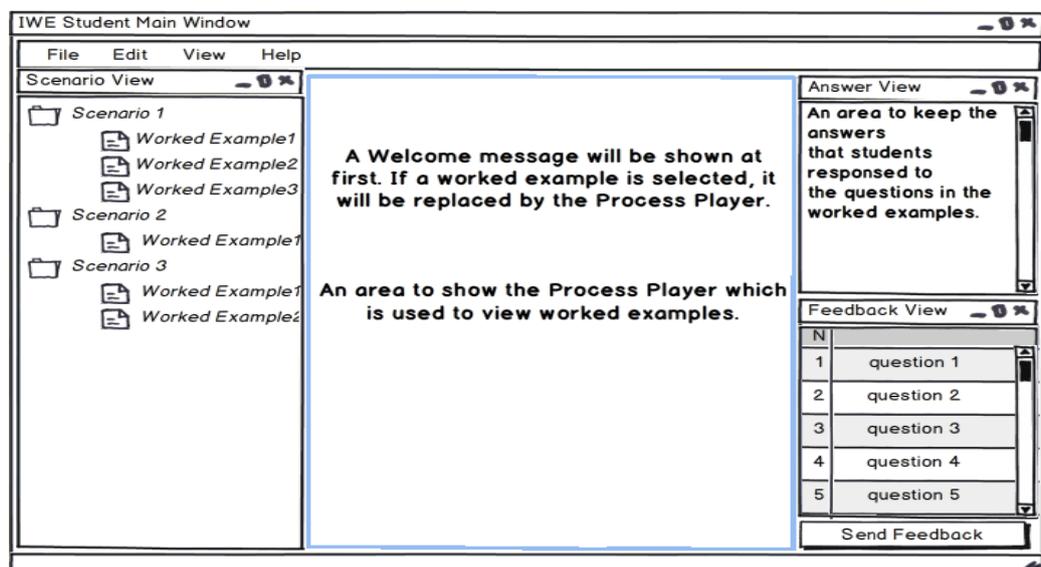


Figure 5.14 Student User Interface- Main Entry Draft Design

It was suggested that the Answer View and the Feedback View should be removed, based on the suggestion of the HCI expert and results of the follow-up one-to-one usability investigation. The reason of doing this modification is to offer more space on the screen for students to explore worked examples, and hide non-related information which may affect the exploration process. The modified design is shown in Figure 5.15.

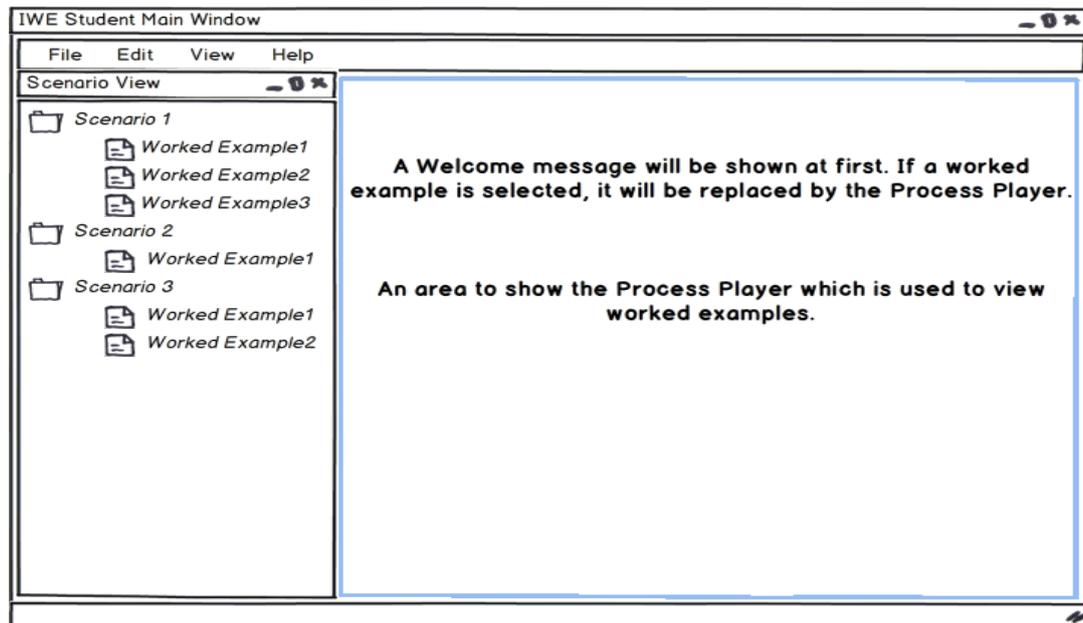


Figure 5.15 Student User Interface- Main Entry Improved Design

The process is the major deliverable by a teacher, so viewing the process is the major task for student learning. Hence, the user interface for exploring a worked example should allow user to fully control the exploration, including start and stop, go back and go forward; and should provide a fixed area to show the explanation of each step.

By following these requirements, a worked example player user interface is designed which is shown in Figure 5.16. Because people are already familiar with using some popular multimedia players, like Window Media Player and iTunes, the design pattern of popular multimedia players is borrowed. A scrollbar is used to change the playing speed, and a set of standard player buttons, like start, pause, next, end and so on are designed on the top of the window. By using these buttons, the student can fully control the exploring procedure. The process player also needs a progress bar which could highlight the current playing status. Due to the player size issue, which is limited to the middle of the main entry, the progress bar is placed at the very bottom of the window. The middle area of the window is used to show the contents of documents with the

predefined structure. For example, Figure 5.17 shows two documents are displayed by using a left-right two panels structure. The Explanation View, above the progress bar at the bottom of the window, is the fixed view to show explanations of each step when a worked example is explored.

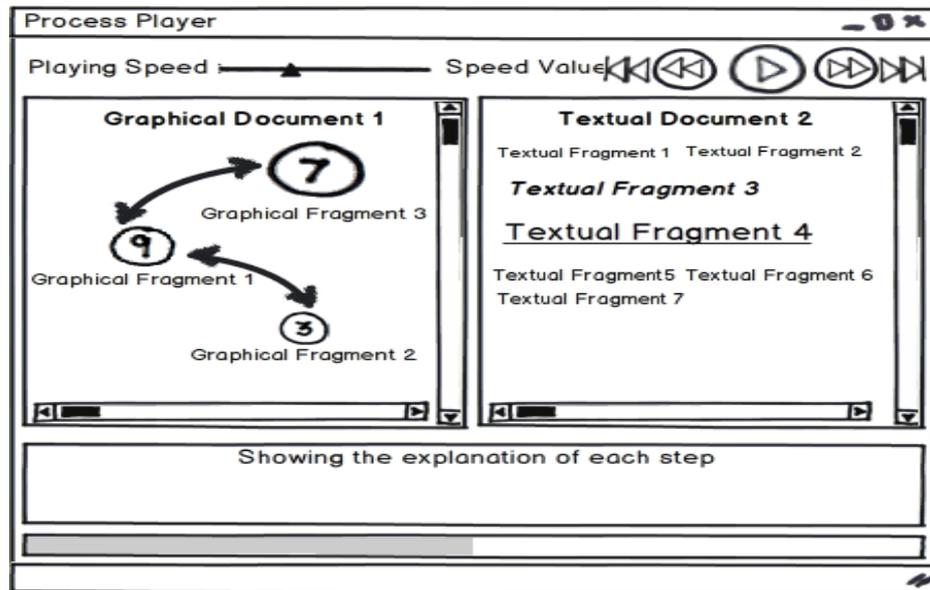


Figure 5.16 Process Play Draft Design

Based on the suggestion of the HCI expert and results of the follow-up one-to-one usability investigation, a “Send Feedback” button is added instead of the previous Feedback View. The “Auto Play” button is removed. The play speed scrollbar and the progress bar are replaced by a step indication slider. Allowing students to control the speed of exploration is also suggested by Tversky et. al. [108]. Hence, the improved design is shown in Figure 5.12.

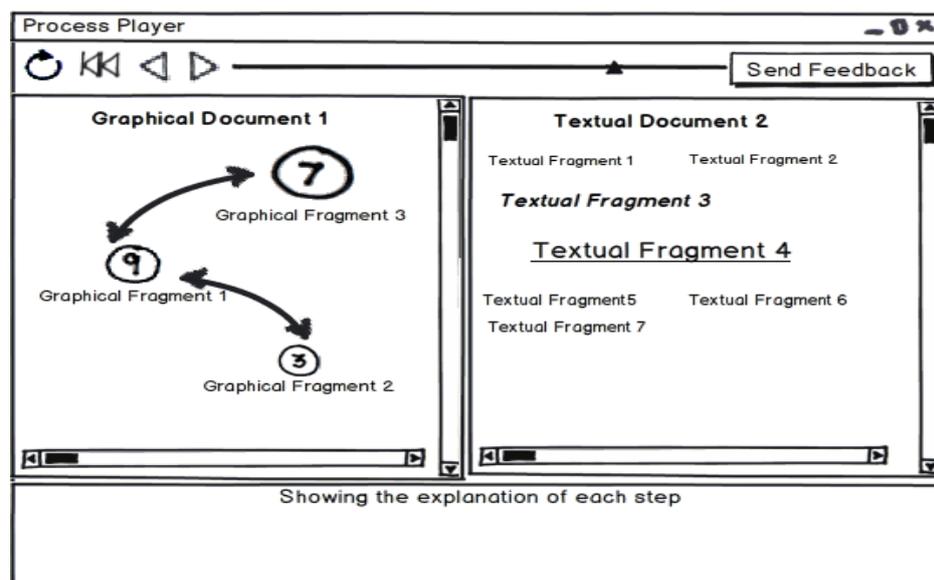


Figure 5.17 Process Play Improved Design

5.2.3 Feedback Mechanism within IWE Design

As discussed in chapter 2 and 4, IWE can be an extra channel for students to communicate with the teacher. Students sending feedback message to the teacher is very important in IWE. When students interact with the worked examples, they can see the teacher's embedded explanations for each individual step in the explanation view, and can send their problems to the teacher. In fact, problems written by a student are saved in a file, which will be sent to a server. The whole sending process is hidden from the students.

The teacher needs to collect these feedback files from the server and then load them into IWE. Hence, the IWE should provide a user interface for teacher to load these files in, summarize these feedback files and generate a new XML file to save this information as course feedback. This could help the teacher to improve the worked example design and collect some learning results of the students. Based on the results of Level-1 Volunteer Students using IWE investigation, it was identified that same mechanism should be applied to students answer files for the embedded questions.

So the feedback summary user interface is designed, which is shown in Figure 5.18 and modified version is shown in Figure 5.19. It contains two parts. The left side uses a tree structure to list all information of the students who sent feedback. The right side uses a tab panel to show an individual student feedback based on selection from the tree node and a collection of all students' feedback messages. The feedback should include the student's information, the feedback given and the name of the worked example. The "Individual Student Feedback Message" tab should summarize all the feedback messages, which are sent by the same student. The "All Feedback Message Summary" tab should summarize all the feedback messages based on the worked example name. The teacher can respond based on this summary information. For example, if many questions came from the same step, the teacher can explain it in the lecture or tutorial; if only a few questions came from different steps, the teacher might reply individually through email or in some other way. The "All Answers Statistics" tab should summarize all the students' answers to the embedded questions, sorted by the worked example name. So the teacher can use this function to observe students learning progress, based on their answers.

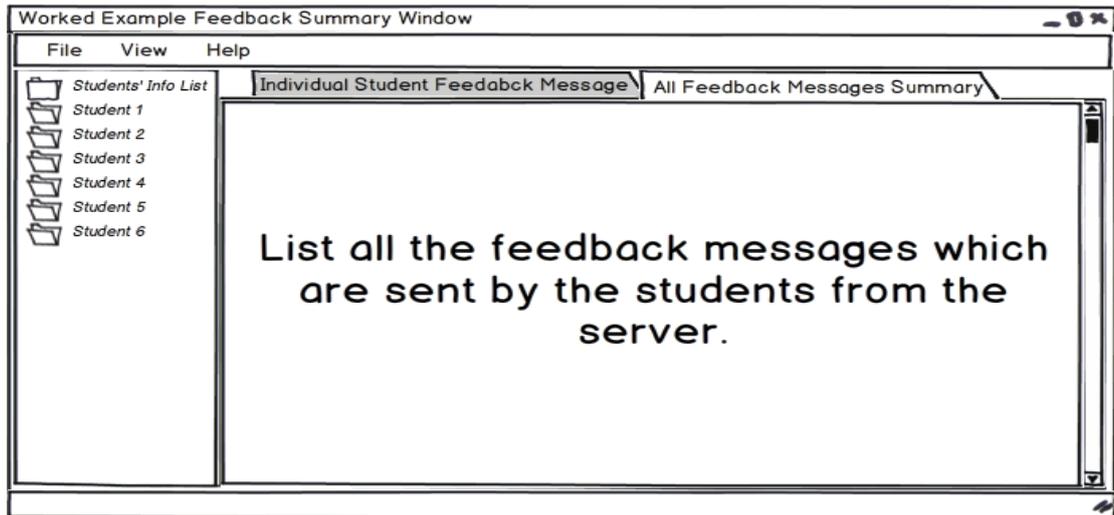


Figure 5.18 Feedback Summary User Interface for Teacher Draft Design

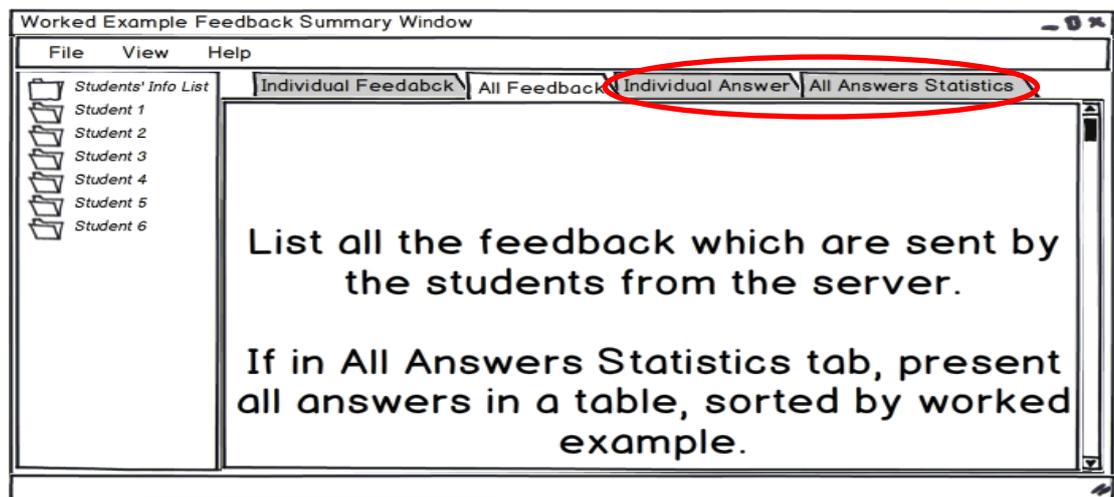


Figure 5.19 Feedback Summary User Interface for Teacher Improved Design

5.3 Final User Interface Implementation

Now these initial designs need to be converted into a practical software program in order to implement and evaluate these designs in practice. The original designs were changed when implemented, due to some suggestions that were collected from the pilot test teachers, HCI expert and user evaluation results. In this section some important user interface implementations will be described in the sequence they are used by teachers and students. For the teacher user, it is required to define document type first, and then to define document. Define an application, and then create processes for presenting different worked examples within this scenario. Later on, collect feedback. For the student user, it is required to build a main entry to select a scenario, and then to view a selected worked example and to write feedback messages. For all users, a standard user interface is required to view the system help contents.

This section contains three sub-sections which list all the final implementation results of the design following different sequences by using screen capture.

- For the teacher user, it follows the creation of process. It also includes the implementation result for collecting feedback.
- For student user, it follows the expected sequence of operations to access worked examples.
- Then the implementation of system help for both users is shown.

5.3.1 Teacher's User Interfaces Implementation

Figure 5.20 shows the first window, when a teacher begins to use IWE. If it is the first time to create worked example, IWE will ask to fill in the course name. IWE will manage different courses separately and keep different teachers' work in individual folders. After creation, the teacher only needs to distribute his own course folder to students. If examples are built for more than one course, the teacher can find requested course through pressing the drop down button beside the course name field. Choose a course from the list, and then click the "Start" button to enter the system.



Figure 5.20 Teacher User's Login Window

Figure 5.21 shows the main entry of teacher's user interface. It contains four views which are application view on the left, middle view for displaying worked examples, document type view on the right top side and document view on the right bottom view. The teacher can access all the creation functions from the tool bar on the top of the GUI. If the teacher wants to edit some existing content, he can right click on a node in one of the side views and then begin to edit or view the content of the selected node.

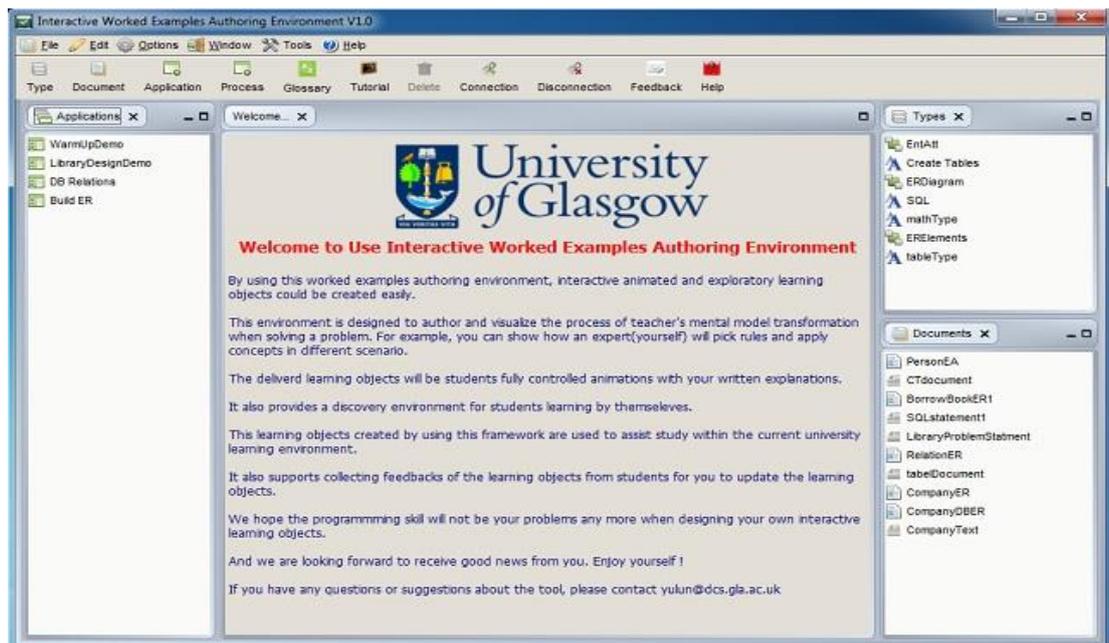


Figure 5.21 Teacher User's Main Entry Window

Figure 5.22 shows the graphical document type editor user interface to describe the target node or line type. The left top area is used to define graphical node type and the left bottom area is used to define line type. The right top area is used to preview the created graphical type and the right bottom area is used to list all the created types. The update buttons in the left side are always disabled, unless an item from the result list in the right side has been selected. This modification is based on a teacher's suggestion, because he may only want

to change one or two attributes values of his design, after previewing the result. Instead of deleting his previous result, he only needs to modify the attribute value and then click “update” button.

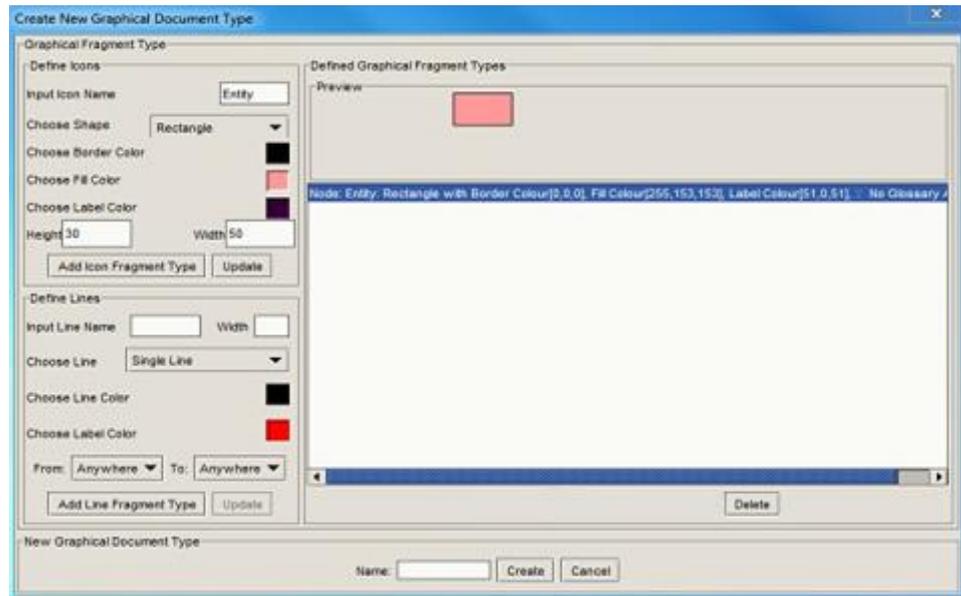


Figure 5.22 Define Graphical Document Type Window

Figure 5.23 shows the textual document type editor user interface to describe the target text styles. The left side is used to setup the attribute values of the text style and the right side used to preview the defining result. It offers a variety of fonts for the teacher user; this allows documents to be formatted so that different components can be distinguished. For example, in a problem description showing examples of inputs and outputs in a different font to the descriptive text. After feedback from teachers, similar to that for the graphical type editor, an “Update” button was added into the window, which allowed modification after previewing the creation instead of deleting the previous work and recreating it.

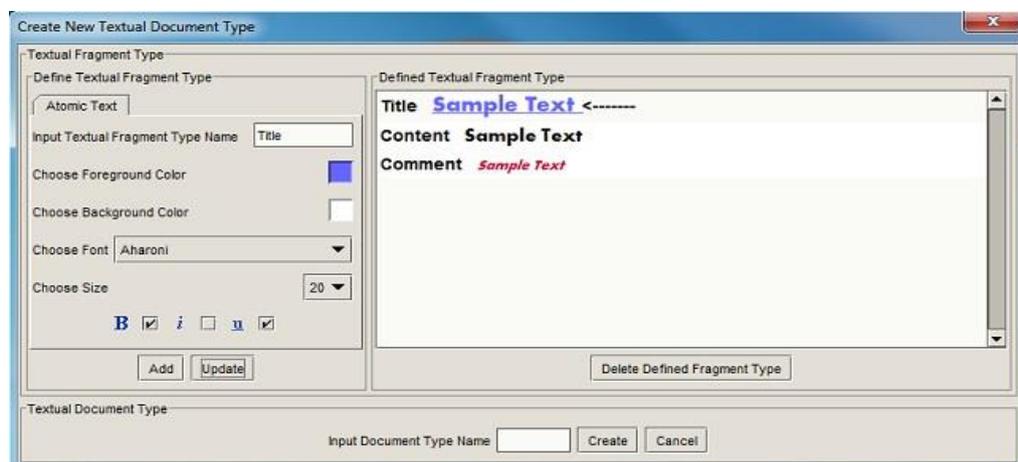


Figure 5.23 Define Textual Document Type Window

Figure 5.24 shows the graphical document editor user interface to describe the target graphical document. The left side top area lists all the node types and the left bottom area lists all the line types based on the selected graphical document type. The right side is the drawing area. Choosing a node type from the list and then clicking a position in the drawing area causes a node of this node type to be drawn at the clicking point. When the mouse pointer is moved inside a graphical node several linking points will be shown automatically. The person_name node in Figure 5.24 shows the linking points for this node. Choosing a line type from the list firstly, then dragging the mouse from one node linking point and dropping the mouse onto another node linking point causes an instance line of the selected line type to be drawn between these two points.

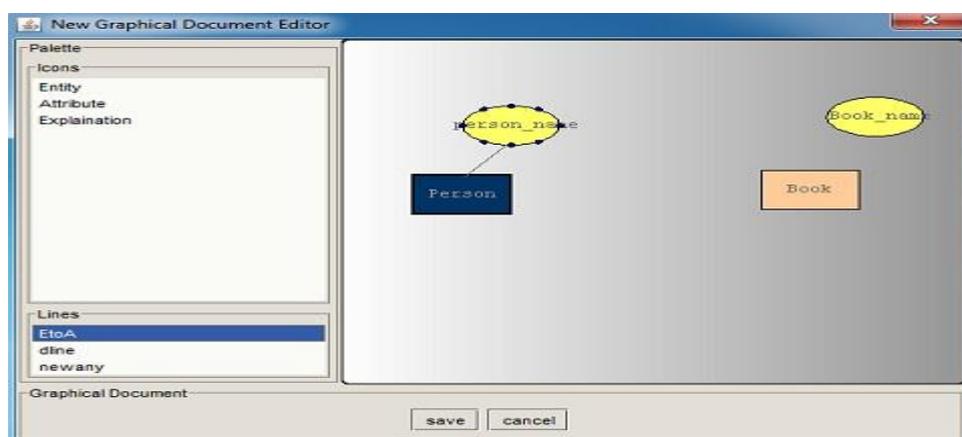


Figure 5.24 Define Graphical Document Window

Figure 5.25 shows the textual document editor user interface to describe the target textual document. The top area is used for inputting the plain text. The teacher can paste text from other resources to this area. These original texts can then be divided into textual fragments by using the “Enter” button from the keyboard. Clicking the “Load All” button causes the system to load each individual line as a separate textual fragment into the table “Value” column automatically. The “Load All” button was not included in the original design. A test teacher suggested this modification after dividing a large plain text, because it could speed up dividing the original texts. Alternatively, the user can select part of the text from the input area by using the mouse, and then click the “Add” button to add a single textual fragment. In order to set the textual fragment type, the user can select the “Type” column in the table to change individual textual fragment type into different styles. Due to the limitation of XML technology used to store documents, the special character “;” is used to represent a “new line” symbol for formatting purposes. This means if the value

of a textual fragment ends with “;”, it will be recognized as the end of a line. So, if the user wants to use “;” as the end of textual fragment, he has to either use double “;” to get a new line or add a space after the “;”. The bottom area is used to preview the editing results.

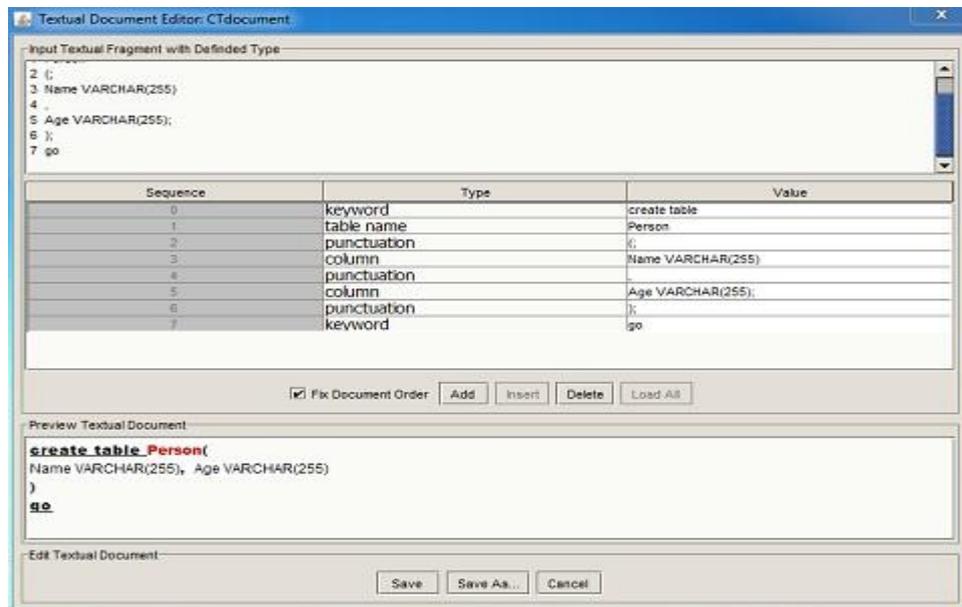


Figure 5.25 Define Textual Document Window

Creating an application is a process of allocating documents into a predefined layout. So a wizard type user interface is implemented. Figure 5.26 shows the user interface to select a predefined layout to be used to present documents. There were 8 layouts at first, however, due to a teacher’s special request to represent a middle stage of solving a problem, another two new layouts, which are the three vertical and horizontal panels, were added in during the final evaluation period. After giving a name for this application (for students it is called scenario) and choosing a layout, the “Next” button will become available. Click on “Next” button to begin allocating documents in the selected layout.

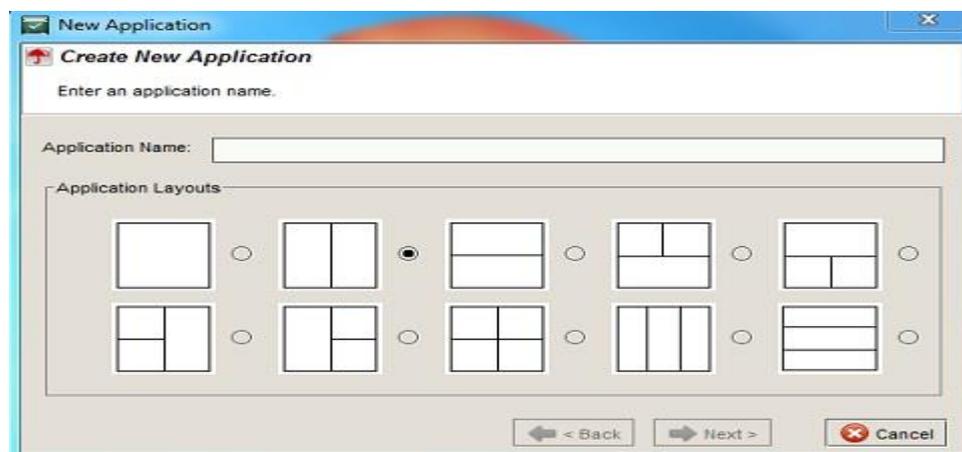


Figure 5.26 Select a Predefined Layout for Creating an Application Window

The nature of the worked example determines the most appropriate layout to use. For example, three panels may suit a programming problem where there are documents for the problem description, a plan, and the program itself. The nature of these documents (a large number of relatively short lines, particularly in the case of the plan and program) suggest that the three documents should be in the vertical format, side by side rather than stacked. Users can change the relative sizes of the panels while exploring a worked example. Figure 5.27 shows an example of selecting a layout of two vertical panels. The user can click a panel to choose a document from a pop up box. And then this document will be placed into this panel. After allocating the target documents into panels, click “finish” button to finish the application creating process.

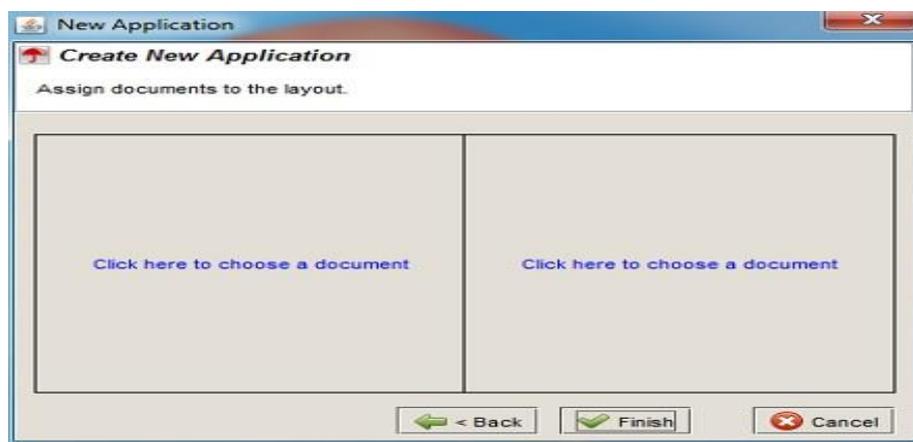


Figure 5.27 Allocate Documents into the Selected Layout for Creating an Application Window

Figure 5.28 shows the result of allocating documents when creating an application in the teacher’s main entry window. In this example, the left panel is allocated a graphical document, which is an ER diagram and the right side panel is allocated a textual document, which are the SQL commands for representing an ER diagram. The teacher can also set up the correspondence between these two documents as an optional choice. Setting up correspondence can be done in the teacher’s main entry window when viewing the application creation result. For example, in Figure 5.28, a correspondence can be set up through selecting entity “Book” in ER diagram and selecting “Book” in SQL commands, and then clicking the “Connection” button in the tool bar. When student clicks one fragment in either “Book” entity in the ER diagram document or “Book” in the SQL document, the other fragment will be automatically highlighted. As mentioned before, setting correspondence is an optional operation for a teacher. If the teacher does not want to set up the

correspondence, he can directly go the next stage to design the process. In this case, a process is going to demonstrate the process of transformation from an ER diagram into SQL commands.

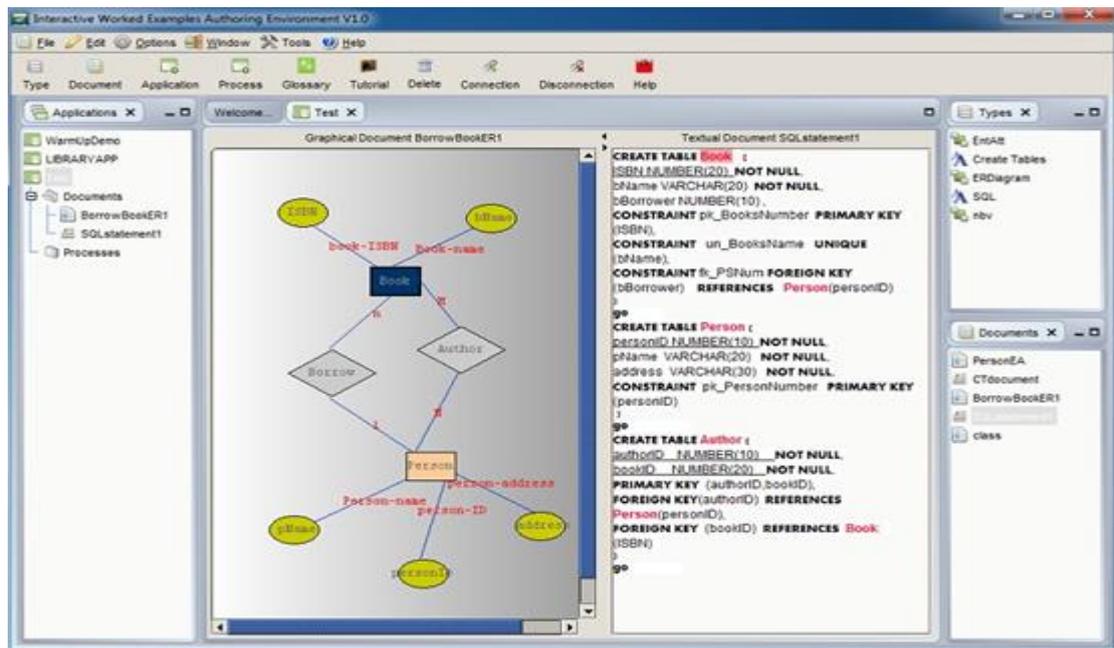


Figure 5.28 Application Creation Result Shown in the Teacher's Main Entry Window

Figure 5.29 shows the user interface to design a process. The left side area is used to show the tree structure of editing results. A Process root node contains step nodes, and a step node contains change nodes and an explanation node. Clicking “New Step” button, a new step node will be added in as the final step. If a step node between other step nodes is selected then clicking “New Step” button will insert a new step node below the selected step node. In order to add a change into a step, a new change can be defined by selecting a document first from the “Documents” combo box, then finding a fragment in the “Fragments” combo box, and then choosing an operation in the “Operations” combo box. To add the change click the “Insert One Change to a Step” button, the new defined change will always be added into the selected step node as a sub node. For instance, in Figure 5.29 the new change will be added into step 12 as a sub node. When a new step node is added in, it contains an explanation sub node. To create an explanation: select this explanation sub node; write explanation text in the text area, which is at the bottom right side; click the “Update Explanation of One Step” button and the explanation text for this step will be added into the step. During the creation of a process, clicking the “Preview” button will show the process player window to the teacher to see current editing results.

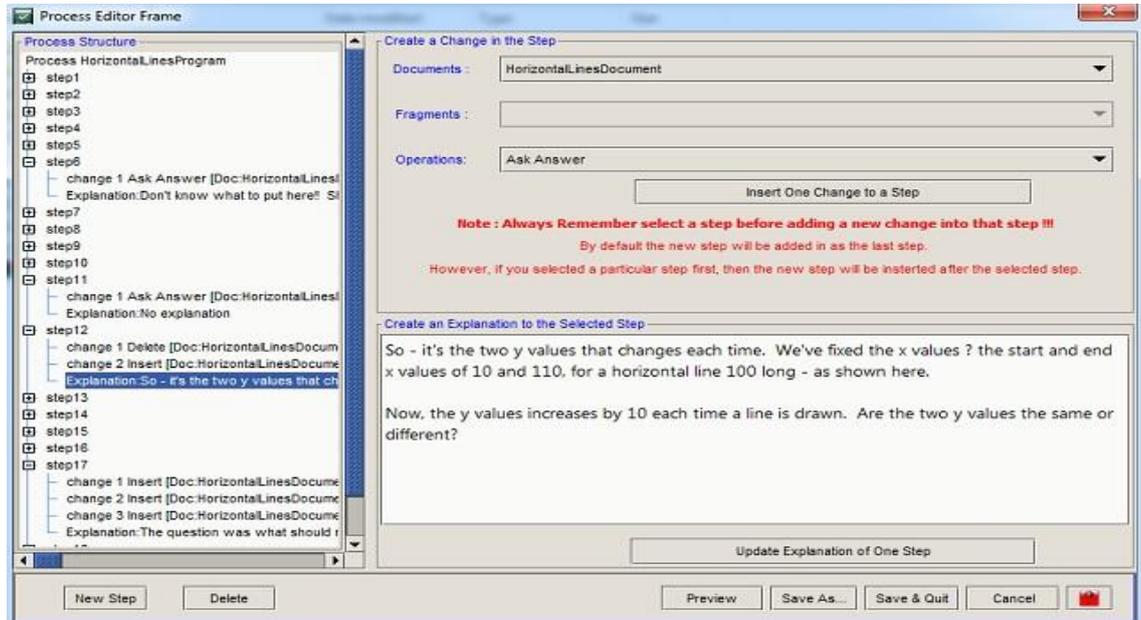


Figure 5.29 Process Editing Window

In Figure 5.30, it shows how to create an embedded multi-choice question as a step. The teacher needs to select “Ask Question” operation from the “Operation” combo box to bring the popup dialog out; and then write question content in the “Question Area” and the options contents in the “Options” Area. It is also possible to create a descriptive question where the student is expected to type in a textual answer; in this case only the question content needs to be entered, there are no options.

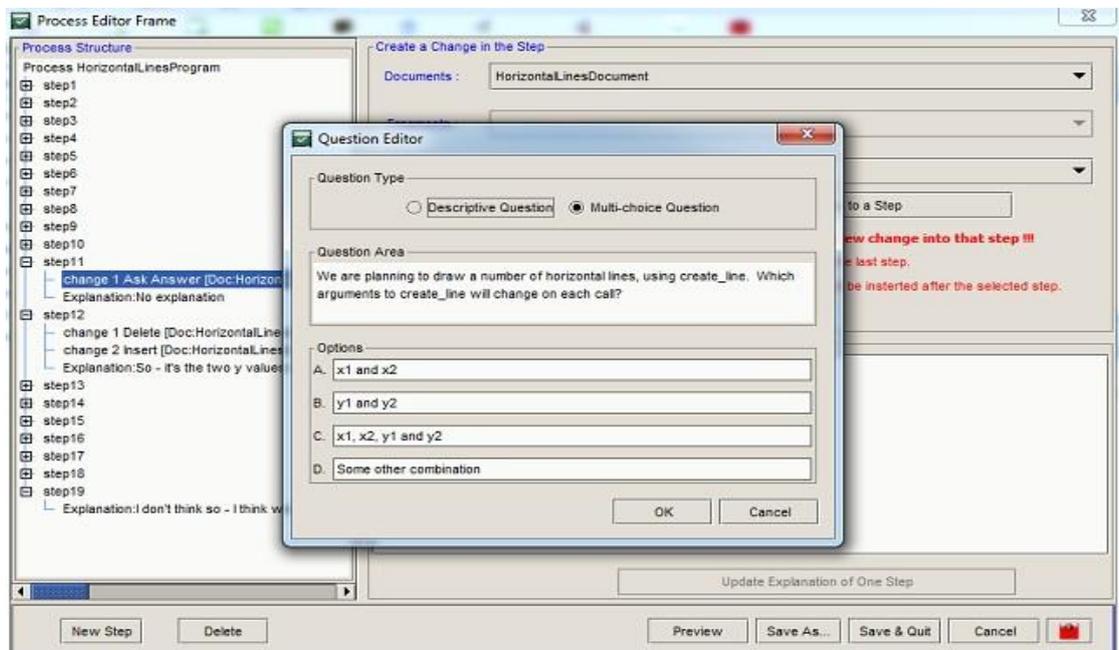


Figure 5.30 Process Editing Window

Figure 5.31 shows the feedback summary user interface. It contains a student’s information area on the left side and a tab panel for viewing different summary

results on the right side. To access this window, the user needs to press the “Tool” option in the menu bar of the teacher’s main entry user interface and then select the “Feedback” menu item. Loading an individual student’s feedback file into IWE can be done through the “File” menu in the top of the window. Clicking on this menu, it will guide the user to select a student’s feedback file on the server and then automatically add and sort the contents into the system. In Figure 5.31, all the feedback messages from students for worked examples designed for the CS1CT course are shown in the “All Feedback” tab. In Figure 5.32, all students’ answers to the embedded questions in these worked examples are summarised, using the “All Answers Statistics” tab.

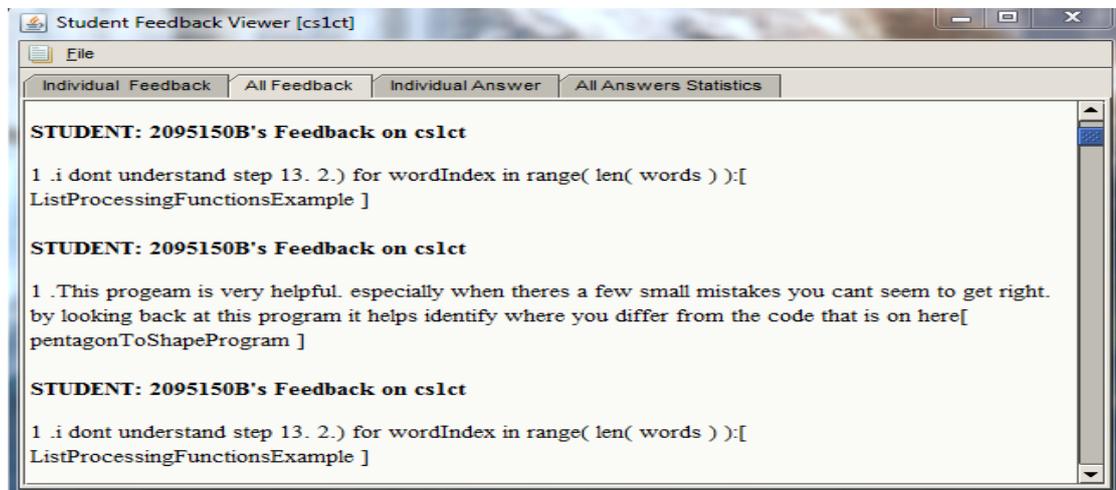


Figure 5.31 Feedback Summary Window

The screenshot shows the 'Student Feedback Viewer [cs1ct]' window with the 'All Answers Statistics' tab selected. The window displays a table summarizing student answers to embedded questions.

Process Name	Process Step	Sum of A ...	Sum of B...	Sum of C ...	Sum of D...	Sum ...	Total Num...	Question
HorizontalLinesProgram	6	11	0	1	0	0	12	Is the solution going to be princ...
HorizontalLinesProgram	11	1	4	6	0	0	11	We are planning to draw a nu...
HorizontalLinesProgram	13	4	6	1	0	0	11	On each individual call to creat...
HorizontalLinesProgram	16	0	5	4	1	0	10	What 3 values should replace t...
RandomLineProgram	4	1	2	0	1	0	4	To control the number of repea...
RandomLineProgram	11	1	0	1	2	0	4	If we ran the code above, wha...
RandomLineProgram	13	1	1	0	2	0	4	So how do we get the segmen...
pentagonToShapeProgram	14	0	2	0	1	0	3	How should the angle be updat...

Figure 5.32 Answers Summary within Feedback Window

5.3.2 Student’s User Interfaces Implementation

Figure 5.33 shows the first window, when a student begins to use IWE. IWE will ask the user to select a course name from the combo box in the middle of the window. After selecting the course name, pressing the “Start” button will cause the IWE to automatically load all the relevant data into the system. A check box is included to meet the ethics requirement if an evaluation is being carried out.

If the user agrees to join the evaluation, the logging function of the system will record all their operations during usage and save a log file onto a server; otherwise, the logging function is disabled.



Figure 5.33 Student Login Window

Figure 5.34 shows the main entry of the student user interface. It contains two views which are scenario view on the left and a worked example view on the right. In Figure 5.34, it shows an example after user choosing CS1CT course, the scenario view lists all the scenarios of this course and within each scenario all the worked examples are also listed.

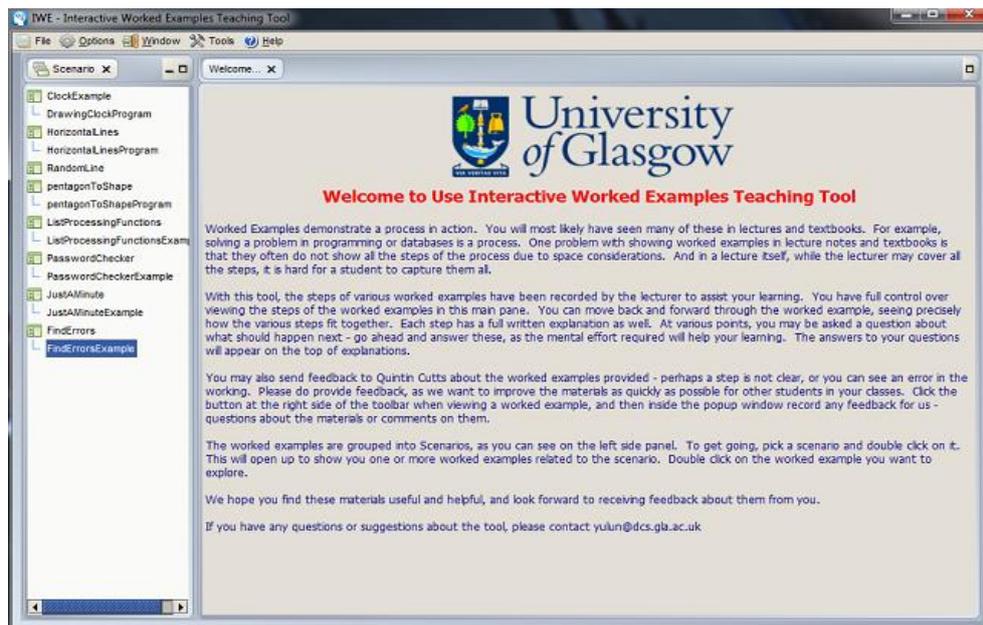


Figure 5.34 Student Main Entry Window

Figure 5.35 shows the process player user interface for viewing worked examples. The top of the player is the control area, which contains several control buttons (roll back, go to beginning, previous step, and next step), a progress step slider for indicating the progress of viewing and quickly accessing any step which has been viewed, and a feedback button. The bottom area is the explanation area, which is used to show the explanation text of each step. The

middle area is used to show a worked example, based on the documents and linkages between them that were defined by the teacher.

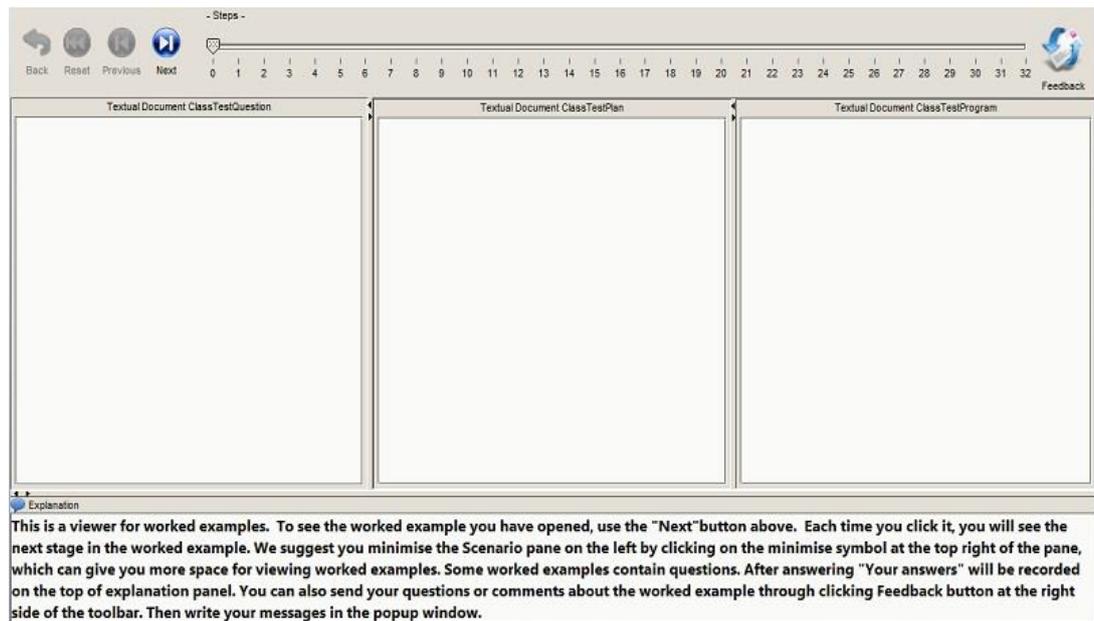


Figure 5.35 Process Player Window

In fact, this process player does not stand-alone. It will be shown in the right side of the student's main entry window. Figure 5.36 shows an example of exploring the worked example "CinemaExPlay" of CinemaExample scenario in the student's main entry window. The operation for student to pick and view a worked example is to double click on a worked example name then the IWE will open the process player in a tab in the student's main entry window. IWE also supports open several worked examples, each individual worked example will be shown in a separate tab.

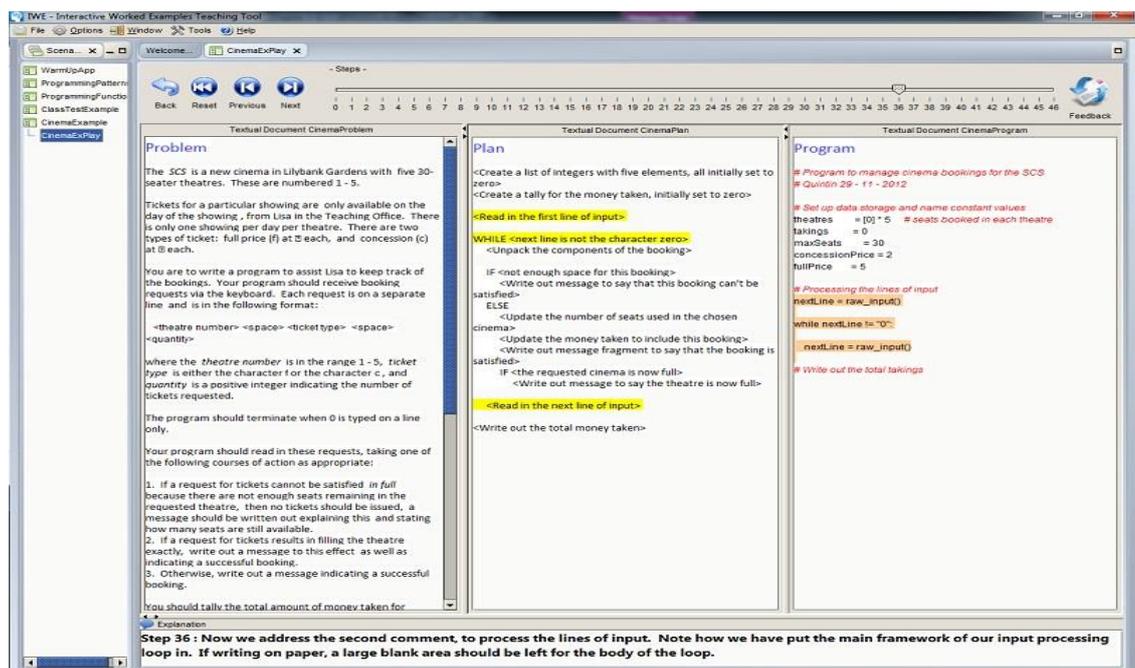


Figure 5.36 Exploring CinemaExPlay Worked Example in the Student's Main Entry Window

Figure 5.37 shows an example of sending feedback, while viewing the worked example. Click the “Feedback” button and a pop up box for inputting feedback messages is shown. For instance, if the user had trouble in understanding step 41; he could write, “I do not understand the explanation for step 41 in cinema example” in the pop up box. Clicking the “Submit” button, this piece of feedback will be sent to the server.

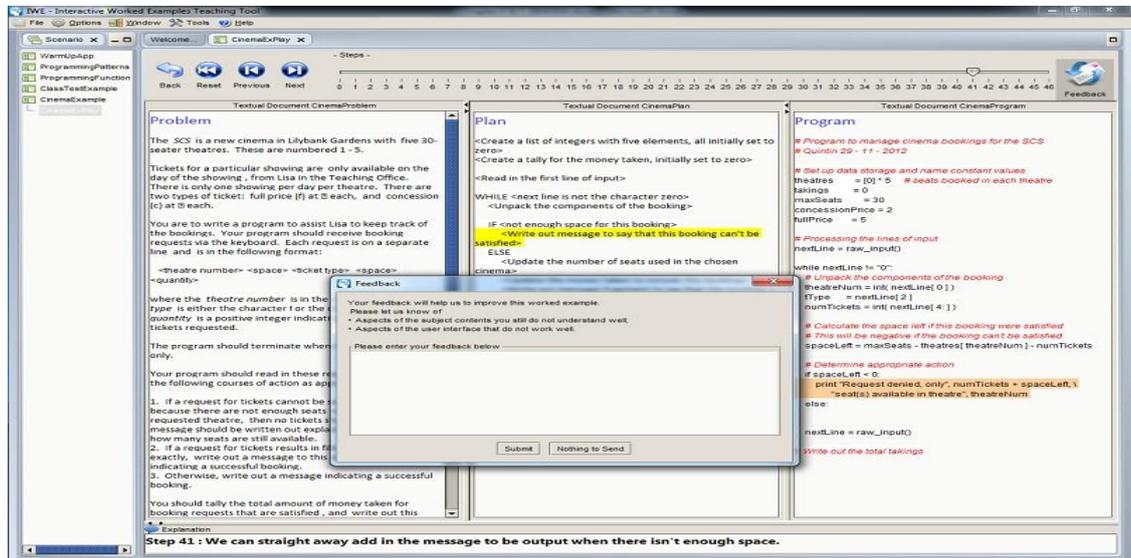


Figure 5.37 Writing Feedback Messages while Running a Process

Figure 5.38 shows the interface for using the correspondence function. It shows a worked example from the CS1Q course, which is designed to help a student understand the process of transforming a requirement description into an ER diagram. Student uses the mouse to click on the relationship “Manage” in the ER diagram, then the related text in the requirement description “started managing” will be highlighted automatically.

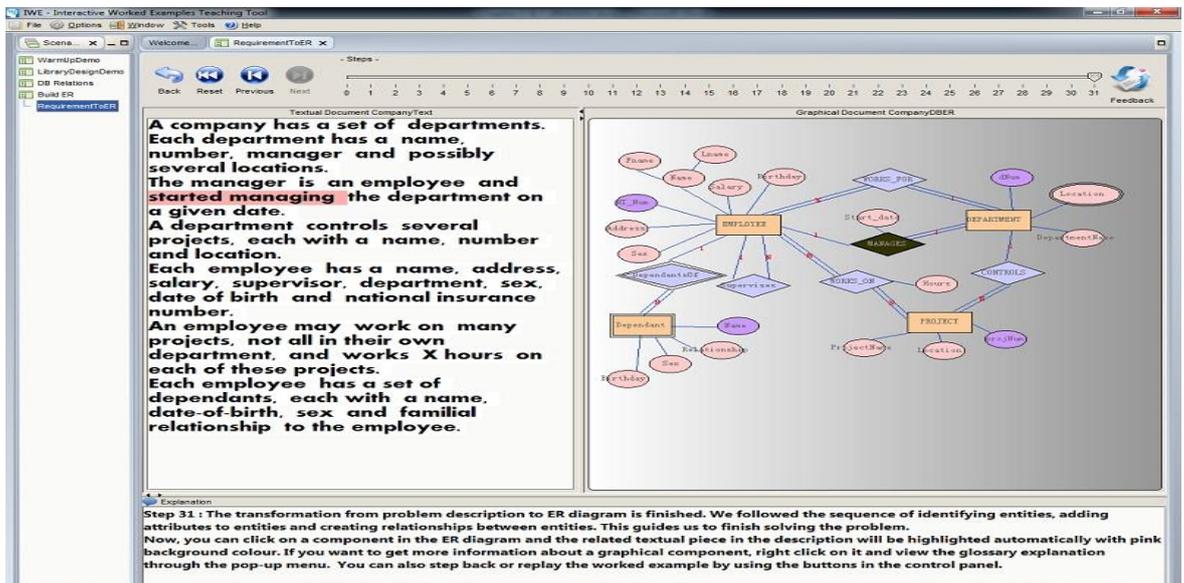


Figure 5.38 Using the Correspondence Function

Figure 5.39 shows the interface for answering embedded a multi-choice question. The students must select an option as “correct” answer, and then click “OK” button. His selection will be recorded by the system in the Answer View, which will be shown in the next step on the top of Explanation View.

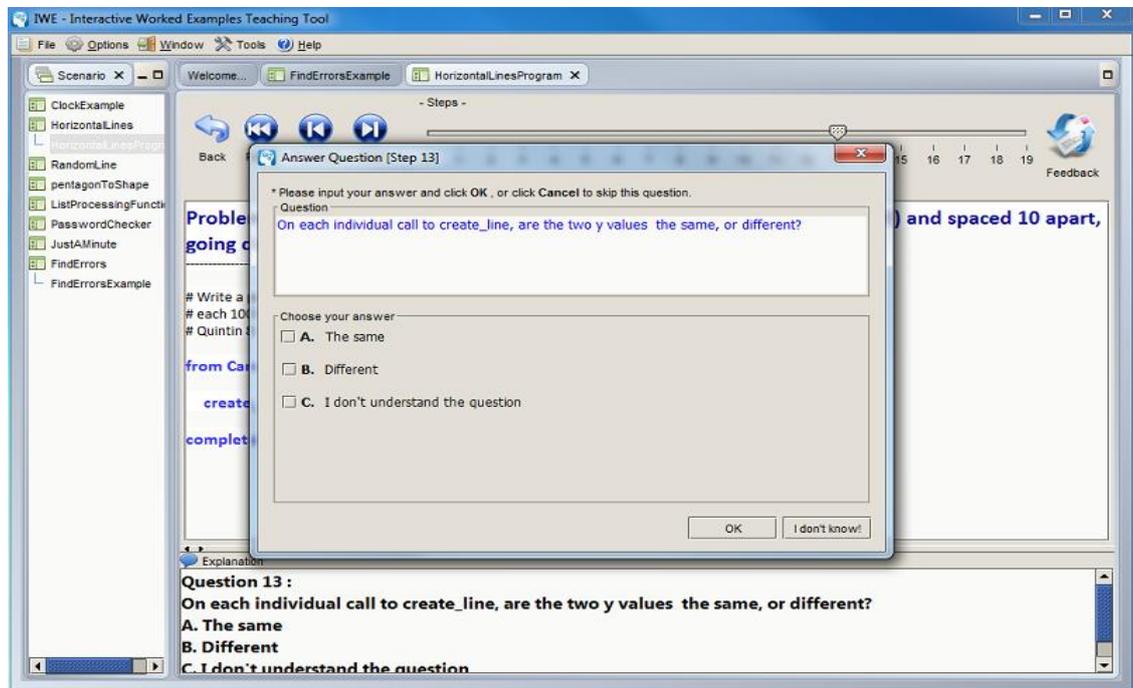


Figure 5.39 Answering An Embedded Multi-Choice Question

Figure 5.40 shows the recorded answer on the top of Explanation View, the student can compare the two answers within this answer step. After this answer step, the Answer View will be hidden automatically.

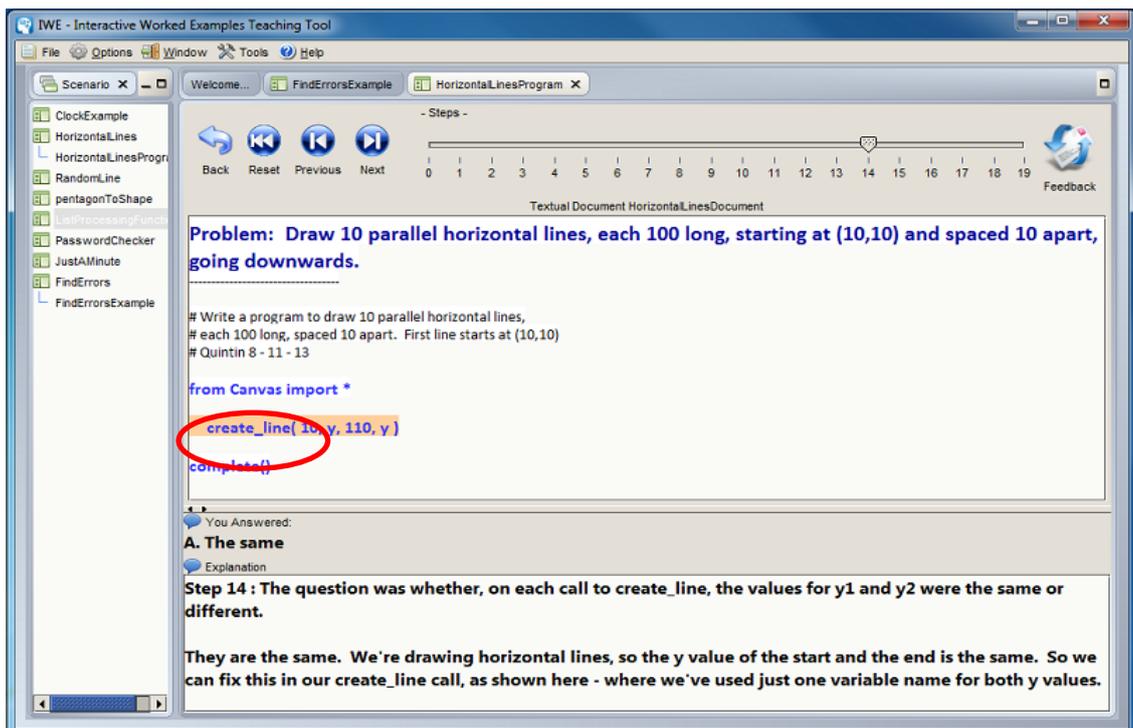


Figure 5.40 Show User's Answer in the Answer View

5.3.3 System Help User Interface

Every system requires a system help function to support its user, if they have problems. Figure 5.41 shows the user interface of the system help. The window is divided into two panels. The left panel is used to list all the titles of help contents and these titles are organized by the functions of the IWE. The right panel is used to show the details of the selected title. The two navigation buttons on the top of the window and in front of the address bar are designed for navigation of the history of browsing the system help functions.

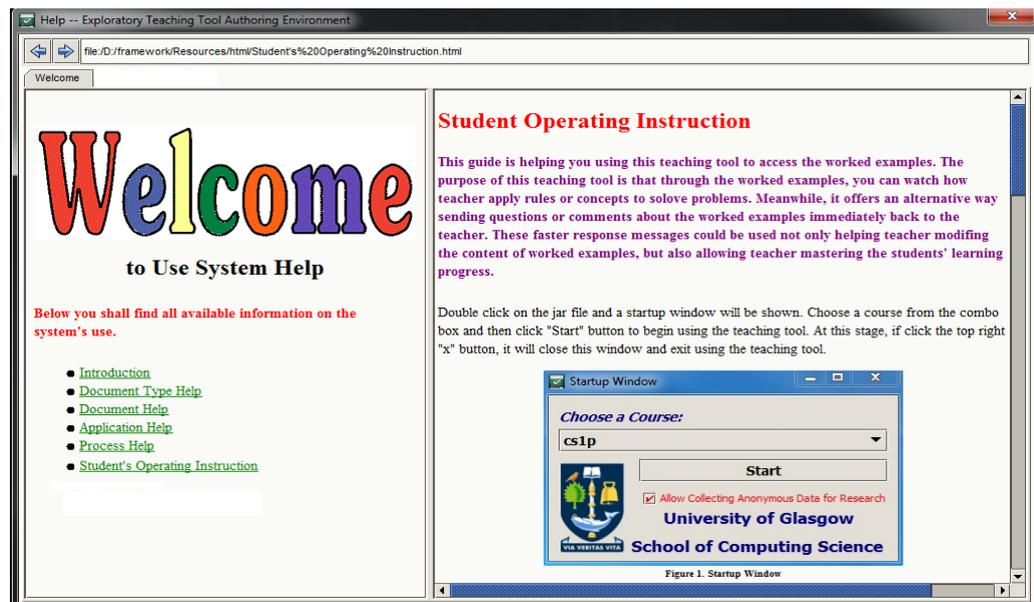


Figure 5.41 System Help Window

5.4 Conclusion

All the major functions of IWE, which are used by teachers to create worked examples and by students to explore and interact with the worked examples, are implemented. Most of the original designs were modified based on evaluation results from the iterative development process. The next chapter describes the sequence of evaluations that were carried out, with students and teachers, which led to the final implementation of IWE described in sections 5.3.

Chapter 6 Evaluations

As described in the previous chapter, the proposed authoring environment for creating interactive worked examples and dealing with students' feedback was designed and implemented. This authoring environment is a proposed tool to help both teachers and students, so it is necessary to evaluate whether it works as expected or not.

The techniques used for evaluating IWE include: survey by questionnaire, interview, observation, expert evaluation, software logging, and collecting user feedback. The process of evaluating IWE was integrated with the iterative development of the software.

Several pilot tests and evaluations were conducted within the University of Glasgow, which aimed either to collect feedback about the user interface, or validate requirements from real end users. Useful information was gathered and successively used to guide or refine the design and development of the tool. A usable prototype of IWE was evaluated with real end users and the results of this were used to further improve IWE for use in a final field study, described in Chapter 7.

6.1 Timeline of Evaluations

The development approach was the well-known software engineering approach of evolutionary prototyping, defined as “an initial working model of the software or part thereof is developed based on the abstract or outline specification. This prototype is evaluated and refined through a number of stages to get the final product [109].”

An initial prototype of IWE was developed based on requirements and examples suggested by Dr. Richard Cooper's experience of using worked examples for teaching database courses. The researcher, also the developer of IWE, used IWE to transform worked examples presented in a text book into interactive and digital format. All worked examples used for evaluations were developed using an initial prototype of the tool. The overall aim of the evaluations was to improve the usability of IWE, for students and teachers, and extend the range of

interactive worked examples, in order to answer the research questions. The timeline of each evaluation during the iterative development process is shown in Table 6.1. The details of each survey or evaluation will be described and discussed later in this chapter.

Section No.	Time	Users	Objectives of Evaluation
6.2.1	15/11/2010	Level-2 Students	Pilot testing of the student user interface of the first version of IWE, focused on identifying the problems of that interface.
6.2.2	15/12/2010 - 16/12/2010	MSc students registered on the ISD course	A survey of learning needs in a database system course to help the researcher to identify the realistic worked examples to be prepared.
6.2.3	26/01/2011 - 07/02/2011	MSc students registered on the ISD course	A feasibility evaluation to identify whether IWE is useable or not by the target users. Three desired worked examples, which focused on transferring ER diagrams into SQL commands, were developed by the researcher using IWE, based on the previous survey. Collecting requirements from real end users was another desired target.
6.2.4	07/06/2011 - 21/06/2011	A teacher who is an expert in the field of software engineering	A pilot testing teacher user interface of the first version of IWE to find out the difficulties experienced by the teacher user while developing a worked example. Modifications were made before inviting other teachers to use it for the educational purpose.
6.3 & 6.4	19/10/2012 - 14/12/2012	Two teachers and volunteer Level-1 students	Qualitative Evaluation of IWE by teachers and quantitative testing of IWE by students to investigate the feasibility of using IWE in a real education context. Teachers developed worked examples using IWE for two different courses, to verify that individual teachers can create interactive worked examples and validate that these examples are usable by the students.
6.5	23/01/2013	An independent HCI expert	Qualitative Evaluation of IWE by Interview to identify problems associated with the design of user interfaces and to guide the later usability investigation.
6.6	26/01/2013 - 28/02/2013	Volunteer MSc students; Level-1 tutors and students.	Follow-up One-to-one usability investigation that aimed to observe and investigate how individual subjects actually used the tool.

Table 6.1 the timeline of evaluations during the iterative development

Steps 1 to 4 were carried out after the first development cycle and focused on testing the initial user interface for both students and teachers. Doing these evaluations ensured that IWE can be used not only the researcher, but also the

target end users - both teachers and students. These four activities aimed to test the user interface during iterative development, so are only briefly described in section 6.2. Because the results affected and guided the further development work, a brief overview of each activity and summary of the important conclusions is given.

Steps 5 to 7 were carried out after the second development cycle. They are described in sections from 6.3 to 6.6. These three evaluations focused on how the end user actually used IWE and found out the problems during usage. Step 5 is described in two parts; the teacher's evaluation is described in section 6.3 and the student's experience is described in section 6.4. Step 6, the independent HCI expert evaluation is described in section 6.5. The investigation process and results of step 7 are reported in section 6.6. The conclusion of the chapter in section 6.7 summarizes the results of these evaluations and identifies further possible improvements to the IWE user interfaces.

Standard ethical procedures were followed for each of the evaluations in this chapter. In particular, referring to Table 6.1:

- 6.2.1 The students were volunteers, taking a course over which neither author nor supervisor had any influence. All data collected was anonymous. Students were briefed and debriefed in the usual way.
- 6.2.2 The students were volunteers, taking a course for which the author was a lab assistant with no assessment responsibilities and over which the supervisor had no influence. All data collected was anonymous and students were briefed and debriefed in the usual way.
- 6.2.3 The student volunteers and the ethical conditions were the same as for 6.2.2.
- 6.2.4 This involved working with a member of the lecturing staff. No ethical considerations involved.
- 6.3 This involved working with two lecturers, to evaluate the development of worked examples. No ethical considerations involved.
- 6.4 Worked examples created as part of 6.3 were offered to students as part of the lecturers' classes. Lecturers were in a position of authority over the students and so a formal ethics application was submitted to the

University and approved. The author was not in a position of influence over the students, handled all data collection, and ensured that lecturing staff saw only fully-anonymised data. The system logged student activities, but students could opt out of this data collection while still being able to use the worked examples to their full extent. Students were introduced to the system and to the mechanism for opting in or out of data collection in a lecture and via email. When the system ran, a dialogue box appeared initially asking the student whether they wished to opt out of having their data collected. Some students were seen to take this option, showing that the process was understandable to them.

- 6.5 This involved working with an HCI expert. No ethical considerations involved.
- 6.6 Continuation of the study in 6.4, where individual students and tutors from the courses involved took part in one-to-one sessions working through worked examples with the author as an observer. All findings presented anonymously. This work covered by the ethics application from 6.4. The participant consent form is included in Appendix 1.

6.2 Initial Evaluations for Testing Purposes

An authoring environment was built to create interactive worked examples. Results of these evaluations include: evaluation of the prototype student user interface; survey of learning needs; follow-up initial feasibility study; and pilot test of teacher user interface; showed that extra development work was needed to facilitate the target users. Collecting suggestions, identifying requirements and validating research ideas from the target users were the major focus in this development cycle.

6.2.1 Evaluation of Prototype Student User Interface

Evaluation of the IWE prototype students' user interface was conducted by using Level-2 students who were taking a Java Programming course. Subjects were asked to fill in a questionnaire after using the tool. In total, 15 subjects evaluated the user interface. 5 subjects only gave oral feedback and 10 subjects completed and returned the questionnaire. Among these 10 subjects, there were 9 male subjects and 1 female subject. 7 had attended a database course or at

least had some knowledge of database systems. 3 had experience of evaluating user interfaces. There was 1 subject who had difficulty distinguishing different colours (colour blindness). The questionnaire contains 17 questions; 8 are open questions and 9 are closed questions. The conclusion of the user interface evaluation was:

- It is crucial to write good instructions to guide users to use the system.
- Make several small modifications to the user interface as suggested from questionnaires, for example, making the “Process” button which was the starting point much more prominent; changing the icon for the “Help” button to make it more obvious.
- Set up default process speed of 3.5 seconds/step instead of 2 seconds/step.

6.2.2 Survey of Learning Needs

6.2.2.1 Survey Process

Due to the researcher’s lack of teaching experience, a survey of learning needs was conducted. This survey was used to collect some problems when observing students using their knowledge to implement practical examples. Identification of these problems allowed the researcher to create explicit targeted worked examples to assist students in understanding related parts of knowledge.

The survey was carried out in the MSc students’ lab from 15/12/2010 to 16/12/2010. At the time they were answering the questionnaire, all the lectures and tutorials were finished. The questionnaire was handed out only 2 days before the deadline of the final assessment, so some of the subjects may have completed their answers with less attention, which may affect the accuracy of the data collected. However, based on a total of 7 hours’ observation in the lab, responding to questions while they were working on the assessment, the researcher had more chance to identify the real problems, which were not included in the survey questionnaire.

The observation of students working on their assessments and the survey results showed students did need extra support for their Information System and Database (ISD) course, especially on using foreign keys to represent different

relationships between entities. Hence, worked examples were created to explain how to add foreign keys to entities to represent the relationships between entities in ER diagrams.

6.2.2.2 Survey Results

- All subjects attended the ISD course in master level. 22 subjects took the survey and filled in the questionnaires.
- 20 out of 22 subjects wanted extra support to the ISD course. The existing lectures and tutorials were definitely not sufficient for subjects to learn and master database knowledge. So a multiple-choice question asked subjects to list their priority of how the extra support could be given was designed. Figure 6.1 shows the result of students' preference of the extra supports. It identified that a teaching tool can be controlled by students under the individual learning pace was much desired. Option 4 may be a leading choice, due to the added explanation about how to use the tool. The reason of adding this piece of explanation was to emphasize that students can control their learning pace. This is a major difference compared with a lecture or other uncontrolled teaching tools.

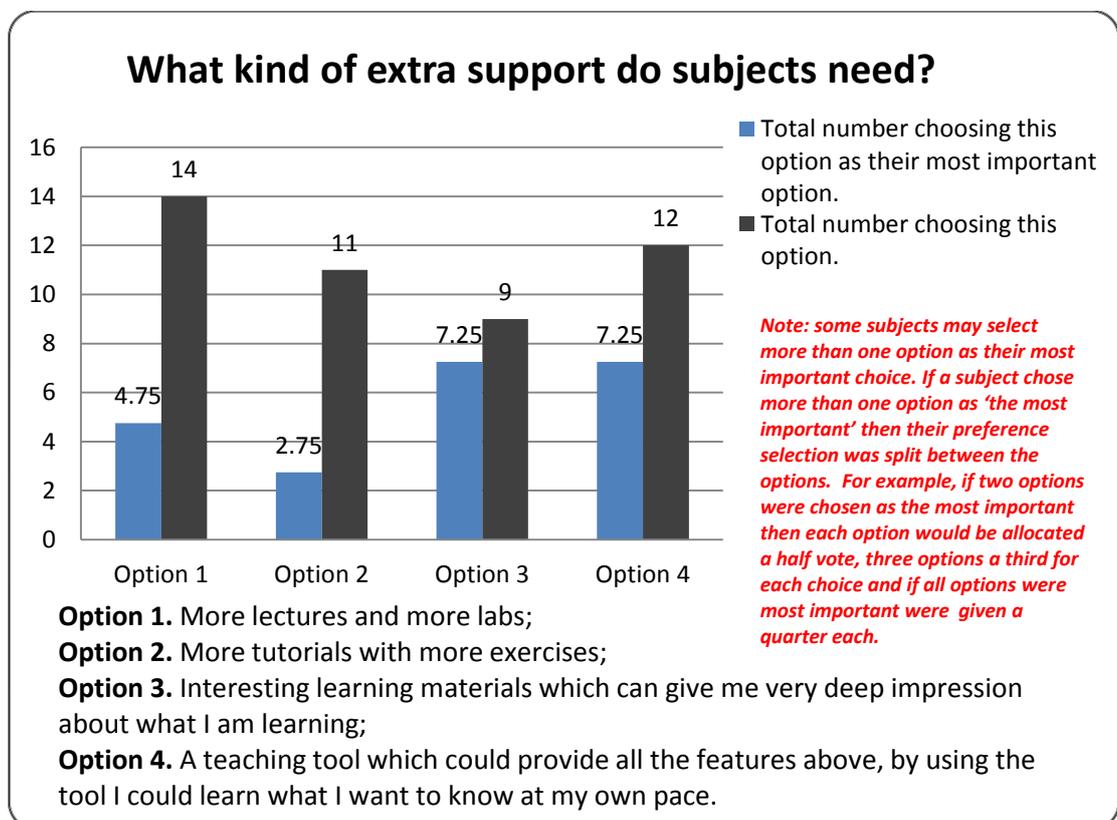


Figure 6.1 The Result of Extra Support Required from Subjects

This survey identified two major issues.

- Subjects wanted extra support while they were learning on their own. They needed more tutorials and exercises in the lab and wanted some instant explanations about what they were doing or when they were doing something wrong.
- Subjects with some previous knowledge about database systems could complete the assessment, but not all of these subjects were confident enough to finish it. Learners, especially novices, do have difficulties applying the knowledge learned about the process of problem solving. For example, they learned the rules for adding foreign keys to represent the relationships between entities in ER diagram; however, when writing SQL commands to implement the design, they confused or even forgot to add foreign keys, or forgot to set up composite primary keys.

Therefore, it was worthwhile to provide transformation-based worked examples to help novices to build their initial experience and then apply suitable rules in the process of problem solving.

6.2.3 Follow-up Initial Feasibility Study

6.2.3.1 Motivation for Design of the Initial Feasibility Study

The feasibility study was designed to identify the advantages of tool when representing worked examples and the problems of the current version of the tool. It aimed to test whether IWE can be used by the target users with the purpose of exploring or interacting with the worked examples. It also asked for suggestions to improve the tool from subjects and collected some real user requirements, which can be considered in the further development and promote the further research.

Based on the survey results related to learning database concepts, three worked examples, which focused on transferring ER diagram into SQL commands, were translated from a database text book by the researcher. Two were designed to explain how to add foreign keys when writing SQL commands for representing 1-to-Many and Many-to-Many relationships in the ER diagram. The other one focused on the entire process of transferring the ER diagram into executable SQL

commands. These three worked examples covered most of the problems, identified from the survey of learning needs and observation of the students' performance of working assessment in the lab.

6.2.3.2 Initial Feasibility Study

This feasibility study was carried out in MSc lab between 26/01/2011 and 07/02/2011. There were 25 subjects in total taking part in this study. 7 subjects were MSc Bioinformatics master students and 18 subjects were MSc Information Technology master students. Most subjects finished their tasks through a one to one scenario except the subjects from Bioinformatics due to time constraints.

Because these subjects met a lot of problems while working on their ISD assessment and the 3 worked examples were designed with the aim of helping to work out these problems in the future, subjects were eager to explore these worked examples by using the tool. The subjects interacted with the 3 worked examples step by step by using the process function provided. Instructional guidance was provided to help with using the tool, based on the evaluation result of prototype student user interface. After exploring the 3 worked examples, a questionnaire to evaluate the tool and the idea of the research project was invited to be filled in. 25 valid evaluation questionnaires were collected, as the subjects all attended the ISD course and submitted their assessments.

6.2.3.3 Results of Initial Feasibility Evaluation

Producing usable interactive worked examples that students could use was validated, as the results of this study showed that subjects were able to use the tool to explore and interact with these 3 worked examples. They liked the idea of providing instant explanations to the worked example step by step which interpret what was happening in a step, and why contents changed in the different panels. However, the way of presenting the instant explanations was not good enough and required to be improved.

Subjects were not only satisfied with the demonstrations of the transformation process of problem solving, but also preferred a more interactive teaching tool, which could provide instant explanations while they were learning. They

believed more interactivity can bring more benefits for their learning. For example, a possible method is to allow subjects to select some contents from one model and the relevant contents in another model should be automatically highlighted to show correspondence or links between the models. Another possible method to increase the interactivity is through providing a question and answer mechanism for students to check their own progress. This mechanism also helps the teacher to build faded worked examples for students to interact with. Another interesting suggestion was to allow subjects to make changes in one model with the relevant contents in another model being automatically modified. However, this is outside the scope of the current tool, which does not support algorithmic transformations between models.

6.2.4 Pilot Test of Teacher User Interface

Previous evaluation results show that usable interactive worked examples can be produced by the researcher using IWE and students were able to use the tool to explore them. The next step is to test whether a teacher user can produce interactive worked examples by using IWE. So a pilot test was conducted. It aimed to identify problems of using teacher user interface to create interactive worked examples in another topic and to be prepared for the teacher's evaluation later on.

6.2.4.1 Motivation

From 07/06/2011 to 21/06/2011, a pilot testing for teacher user interface was conducted by an experienced professor, Ray Welland, who taught software engineering courses for many years. He transferred an existing worked example, which used in the software engineering course of the MSc IT into IWE. The example demonstrated how to draw a use case diagram from a requirements specification. Creating use case diagrams is a typical transformation-based problem in the computing science subject, as there are guidelines that can be followed for identifying users and use cases, however, making the judgments relies on the previous experience. As the expert worked in software engineering subjects for so many years, he can give suggestions based on his rich experience of building and testing software. So he is an ideal teacher user for the pilot test.

6.2.4.2 Results of Pilot Test of Teacher User Interface

Based on the expert pilot test, the test version was not efficient enough for inputting text contents and not convenient enough for editing and formatting text, especially for splitting original plain text into textual fragments.

The adjustment made for this suggestion is to allow the user to copy and paste from other existing resources and then divide the original text document into a set of textual fragments by using the “Enter” button on the keyboard. A new textual document for IWE using will be automatic created with multiple textual fragments. As the teacher already had a collection of worked examples in documents, this could speed up creating worked examples.

Three other significant problems related to the creation of processes were identified, which were related to creating explanations for a step. Problems and suggested solutions are described below:

1. Explanation of each step is important, but creating a separate textual document to keep all the explanations for each individual step is cumbersome. Later on if modification is required, it is even harder to use. It was quite hard for the teacher to prepare a proper textual document before creating a process. In most cases, the explanation for a step is unique. If creating several processes, the teacher needs to prepare all the explanations for different processes in a single explanation document, which was even harder and unusual.

The adjustment made for this suggestion is to integrate the step explanation text as a part of a step, which means the explanation should be saved as a part of step in the process XML file. The explanation text should be added in while defining a step of a process.

2. In the process editor user interface, it needs to provide facility for adding the specific explanation to a step.

The adjustment made for this suggestion is to add an area for user writing the specific explanation to a step in the process editor user interface. Modify the tree structure of representing process editing results. Add an explanation node

as a sub node of a step node in the process tree structure and each step only contains a single explanation node in the process editor.

3. The explanation text should be always in the same place in the student user interface while they are interacting with the worked examples.

The adjustment made for this suggestion is to add a textual panel for displaying the explanation of each step in the bottom of student user interface.

6.3 Teacher Evaluation

One aspect of the research is that interactive worked examples can be developed and evolved over time by individual teachers using an authoring environment rather than requiring bespoke programming. Based on the previous students' evaluation and teacher's pilot testing results, the second development cycle was finished. A new version IWE was released and ready to be evaluated again by suitable teachers.

6.3.1 Overview the Process of Teacher Evaluation

As mentioned in the usability review section, a usable user interface for highly motivated teachers to achieve their objective is a reasonable result. Although the second development cycle was finished, the IWE was still a prototype, so at this stage a qualitative evaluation focus on using experience and identifying problems is better than a quantitative evaluation focus on measuring satisfaction of user experience. The aim of this evaluation was to answer the following questions:

1. Can the subjects understand the underlying model of how worked examples and their component parts are structured in IWE?
2. Can subjects create their target worked examples by themselves, with the availability of the help contents?
3. Would subjects like to use these worked examples, or even create some additional worked examples for their teaching?
4. Do subjects like or prefer to use IWE to create their own worked examples, compared with the tool they usually use to create worked examples?
5. Would subjects like to recommend IWE to other teachers?

6. Can subjects create worked examples in a reasonable time?

The process of doing this evaluation is described below:

1. Demonstrate a simple interactive learning object, which can be created by IWE using the student's interface. And then give a short training to the subjects to familiarise them with IWE. This short training will include five aspects which are:
 - Illustrate the concepts and terminology which are used by IWE and introduce the target of IWE;
 - Explain the work flow by demonstrating how a learning object is built by using the teacher's interface;
 - Emphasize help content can be used during development of their work;
 - Emphasize the advantage of the IWE authoring environment, including: ease of making modifications to worked examples and support for collecting questions or comments about the worked examples from students.
 - Answer subjects' questions about the introduction and demonstration of IWE until they understand what IWE can do for them.
2. Ask the subjects to create their target worked example or improve one, which they previously created by themselves. Leave them alone for a period and only provide extra help, if subject needs it.
3. Interview the subjects about their experience of creating worked examples using IWE.

6.3.2 Built Examples

Prof. Quintin Cutts, who taught CS1P (Computing Science Level-1 Programming) course, and Dr Marilyn McGee-Lennon, who taught the database part of CS1Q (Computing System Level-1) course, were invited to use IWE to create worked examples for students to use as part of their courses. 10 examples were created by the 2 teachers within the academic teaching semester. 4 of them were designed for the CS1P course and 6 for the CS1Q course.

The 4 examples created for CS1P course are listed below:

- *Unit 7-Task 3-1* example was designed to show how to apply programming patterns to solve a problem;
- *Unit 8-Task 2* was designed to show how to program a function;
- *ClassTestRun* was designed to show a similar question to those students would be expected to solve in the class exam.
- *CinemaExPlay* was an example based on a previous class exam question. This example was designed to show how a plan was created based on the problem description and then how the code was produced based on the plan.

The 6 examples created for CS1Q course were listed below:

- *DB Relations* scenario contains 4 different examples. All these examples are created based on the two documents, one graphical document for representing ER diagram and one textual document for representing tables. The *FromERToTables* example used the whole of these two documents. The other 3 worked examples only presented the relevant partial documents for different educational purposes. These examples are described below:
 - a. *1-to-1 Relation Demo* example was designed to explain how to add a foreign key in a table to represent a 1-to-1 relationship between entities in the ER diagram.
 - b. *1-to-Many Relation Demo* example was designed to explain how to add a foreign key in a table to represent 1-to-many relationship between entities in the ER diagram.
 - c. *Many-to-Many Relation Demo* example was designed to explain how to add foreign keys in tables to represent many-to-many relationship between entities in the ER diagram.
 - d. *FromERToTables* example was designed to explain how to transfer a complete ER diagram into tables.
- *TransferDemo* was a simple example designed to explain how to design an ER diagram from a requirement description.
- *RequirementToER* was a complex example designed to explain how to design an ER diagram from a requirement description.

6.3.3 Experience of Building Examples

Two teachers successfully created a set of worked examples, which directly fitted their courses at the teaching point researched. These practical examples were built within their busy timetables during the academic teaching period. Their detailed experience of using IWE can be found in Appendix 7. Their experience of using IWE can be summarized below:

1. They believed that IWE was a tool for structuring existing documents to be represented as interactive worked examples with a sequence of steps and integrated explanations. The starting point of using IWE was to define document types or styles, rather than creating the contents as well as styling and structuring, as IWE assumed the contents creation were completed outside the tool. Both of them were confused about the sequence of defining document types at the beginning of using IWE.
2. They found organizing a textual document as a sequence of textual fragments was non-intuitive. They also felt that applying fine-grained styles to different textual fragments was unusual compared with other text editors. The way of forcing a new line to format the textual document was unfamiliar. They pointed out that updating the textual documents should be more flexible, as the process was defined by using the textual documents. Building a graphical document was easy to appreciate, but lack of layering of components was a limitation for constructing graphical documents.
3. They required additional options for formatting the explanations and preferred to add hyperlinks to other explanatory materials in the explanation panel. They required even more flexible layouts of the panels to display documents, for example, a mechanism for changing the default size of each panel in different steps. They needed multiple colours for highlighting different fragments, which could support focus on a particular fragment within a highlighted group of fragments.
4. When defining a process, they found no mechanism for selecting a group of fragments and then giving an operation to the group. If a grouping fragments mechanism were added, it could reduce the number of operations on fragments significantly. Very often, they forgot to press “Update” button to modify the explanation text of a step.

5. Although they did not receive significant feedback from students to update the worked examples, they extensively checked and reviewed their creations before delivering them to the students. These worked examples evolved based on teachers' own critical feedback.
6. They identified a useful feature of IWE, which was to represent evolving documents. This feature was not initially anticipated by the teacher or the researcher. As the teacher wanted to model the development of a plan to solve the problem, instead of creating multiple plan documents, one for each refinement of the plan, only a single document that contains all these versions of plan was required.
7. They compared IWE with PowerPoint, a popular worked examples creation tool and concluded that PowerPoint was used effectively for small-scale worked examples. However the complex interactive worked examples created by using IWE cannot or are extremely difficult to create using PowerPoint.
8. They realised that the creation of a worked example required much more intellectual effort than structuring them, whether on paper, in PowerPoint or using IWE. The overall experience of using IWE to structure the worked examples was satisfactory, as transforming an existing worked example into IWE was a straightforward task. However, the teacher who did the pilot test of IWE suggested that past exam questions can become ideal worked examples resources to be used for IWE. This suggestion is valuable, especially for reviewing purposes when students are preparing for their written exams.

6.3.4 Conclusions

The target of teacher evaluation was to try to answer the questions, listed in section 6.3.1. So, a group interview to ask these questions as a part of evaluation process was carried out. The answers are summarized below:

- Teachers can understand the underlying model used to structure worked examples in IWE. Unfortunately, such understanding is not immediately intuitive. It requires subjects to engage in an initial learning phase.
- Subjects can create their target worked examples by using the help contents provided in IWE.

- Teachers did create some usable worked examples and would like to use IWE to create more of this type of worked example.
- Teachers preferred to use IWE to create more complex worked examples, containing more details and explanations, to be distributed to students after lectures. However, for simple worked examples used in lectures, they still preferred familiar tools, like Microsoft PowerPoint.
- Teachers would recommend IWE to other teachers.
- Teachers could use IWE to create worked examples in a reasonable time period. They found the difficult part was the designing of the target worked examples, but implementing them in IWE was straightforward.

6.4 Level-1 Volunteer Students Survey Using IWE

6.4.1 Motivation

This evaluation aims to validate that students can use these delivered worked examples to assist their self-learning. Hence, it is necessary to investigate:

- How these worked examples are actually used by students, while doing self-learning?
- How these worked examples can potentially be modified based on students' experience of using them?
- How the student user interface can be improved to minimize the Extraneous Cognitive Load?

6.4.2 Overview of the Design of the Survey

This survey was designed to ask the level-1 students to use the IWE in a real learning scenario. Examples were delivered during the first semester to students as extra learning materials and they were asked to fill in an online questionnaire voluntarily at the beginning of the second semester.

The IWE was deployed in the Level-1 computing lab. There was a shortcut icon named "IWE" on the desktop screen after they logged in to use the computer. There was also a hyperlink published on the course webpage, so students could download IWE to the own machine and use it in their spare time. The CS1Q and CS1P lecturers announced the availability of IWE in their lectures and

encouraged students to use IWE in the lab as extra support for their self-learning. A simple guidance document of how to use IWE was also provided to all the students.

The survey was organized through two courses and lasted for about 8 weeks. One warm-up example was delivered for demonstration and familiarity purposes and was included in both courses. Another six examples related to the CS1Q course were delivered together, because those examples were designed for the database part of the course. At that time, most of the lectures were finished and students were focussing on finishing the final database assessment. Most of the students viewed these CS1Q examples between 19/10/2012 and 29/10/2012. Another four examples related to the CS1P course were delivered individually from 05/11/2012 to 14/12/2012. The first three examples were delivered weekly and the last one was delivered in the final week of the semester.

The survey results were collected in two ways. One was via the log file created by the tool. If the subject agreed to take part in the IWE survey, the tool kept some operating data for each individual user and saved all of them on a server. The log file contained the user login, the time of viewing the selected examples, the operations of viewing the selected examples and the answers to the questions, any feedback messages they sent, and other operations using other functions of the IWE. Other results were collected using the online survey website Survey Monkey [110]. Students who used the IWE and also agreed to take part in the survey were invited to fill in the questionnaire online. The questionnaire focused on general opinions about the functions of the tool and problems that were found during the use of IWE.

6.4.3 Results of IWE Using Survey

176 level-1 students, registered on the CS1P and CS1Q courses were recommended to use IWE. 85 students agreed to take part in the survey in the lab and their usage data were logged. Using the logging information, it is possible to match the subjects with their class exam results, which were held at the end of the first semester. Based on matching those two results, it indicated that these 85 subjects were well distributed, covering students with very high exam scores to those with very low scores. 54 subjects used the examples

designed for the CS1P course, 40 subjects used the examples designed for the CS1Q course and 9 subjects used examples for both modules. 40 out of 85 subjects filled in the online questionnaire. The results of the survey were summarized below, based on those log files findings and data from the on-line questionnaire.

6.4.3.1 Log File Statistics

From the log files statistics, it showed 85 subjects viewed different examples effectively 121 times in the lab. If an example was viewed for less than 20 seconds, it was not counted, except the *WarmUp Demo* example; since we could conclude they just glanced over the example rather than learning from it. In total 11 examples were provided. 5 examples containing embedded questions, 3 were in the CS1P course and 2 were in the CS1Q course. 21 out of 85 (25%) subjects used IWE more than once; 7 of these looked at CS1Q examples and 14 of them looked at the CS1P examples. 9 (11%) subjects looked at both CS1P and CS1Q examples. Times of each example viewed and the average time subjects spent on examples are listed in Table 6.2. CinemaExPlay is the biggest example; therefore it is expected to have a much higher average view time; the number of views is also small and may not be a representative sample of CS1P users.

Example Name	Course Name	View Times	Average Viewing Time (Seconds)	Embedded Questions
WarmUp Demo	Demonstration	21	80	No
Unit 7-Task 3-I	CS1P	19	282	No
Unit 8-Task 2	CS1P	19	464	Yes
ClassTestRun	CS1P	18	671	Yes
CinemaExPlay	CS1P	4	1145	Yes
1-to-1 Relation Demo	CS1Q	15	143	No
1-to-Many Relation Demo	CS1Q	4	41	No
Many-to-Many Relation Demo	CS1Q	3	55	No
FromERToTables	CS1Q	8	193	Yes
RequirementToER	CS1Q	8	60	Yes
TransferDemo	CS1Q	1	47	No

Table 6.2 Example Usage Statistics Results from Log Files

The use of the worked examples was also analysed by looking at how the students used the different play modes: auto play mode, manual control mode or a mixture of the two modes. The survey of learning needs showed that students preferred a teaching tool that allowed them to learn at their own pace.

So it is worth analysing how students controlled their exploration of examples. Table 6.3 lists this result.

Example Name	Auto Play Mode	Manual Control Mode	Mixed Mode
WarmUp Demo	11	8	2
Unit 7-Task 3-I	12	2	5
Unit 8-Task 2	11	6	2
ClassTestRun	6	10	3
CinemaExPlay	1	1	1
1-to-1 Relation Demo	12	3	1
1-to-Many Relation Demo	2	2	N/A
Many-to-Many Relation Demo	N/A	3	N/A
FromERToTables	5	1	2
RequirementToER	7	2	1
TransferDemo	N/A	N/A	N/A
	67	38	16

Table 6.3 Play Mode Statistics Results from Log Files

Two disconcerting results were found from the analysis of log files. One is of those 14 subjects who used the feedback function; only 4 subjects sent feedback messages to the teacher. So, only 5% of the total users gave feedback, which means that the present method of collecting feedback is ineffective. The approach is passive; users are expected to take the initiative and find the feedback button. It is necessary to find out why subjects did not use this function as expected. One possible problem is that the feedback icon is poorly positioned in the user interface and users did not notice it. More feedback might be obtained if an active approach was used, such as asking students to give feedback when they exit from using a worked example.

The other one concerns users' answers to the embedded questions of the examples. The examples containing embedded questions were viewed 57 times in total. However, there were 28 subjects who gave the answer "I do not know the answer" to every question, which meant subjects clicked the "Cancel" button every time when the answer popup dialog box was shown. The rest of the subjects answered at least one question. Therefore, it is necessary to find out the reason for subjects ignoring answering questions.

Some subjects' answers still can give some clues to their learning results. For example, the "ClassTestRun" worked example contains 7 questions within 32 steps, which are all guiding questions to promote linking previous knowledge or

making decisions. To answer this kind of question requires students to input more text, because the answer may be a sentence. This could be a possible reason for getting “I do not know the answer.” The *Unit 8-Task 2* worked example contains 5 questions within 22 steps, which are all quiz questions. These questions do not require the input of too much text, because the answer to the question could be limited to two words. So, more subjects are likely to respond to these questions. Table 6.4 shows the logging results of several subjects’ answers to the questions in *Unit8-task 2*.

	S1	S2	S3	S4	S5	S6	S7	S8
Answer to Q1	while	while	for	not know	not know	while	while	while
Answer to Q2	def	not know	not know	not know	not know	def	total ()	def
Answer to Q3	average	count_3 9	sumOf	total	Sum	sum	total	total
Answer to Q4	():	()	()	():	()	parameter	not know	()
Answer to Q5	4	not know	loop	index	not know	not know	loop	not know

Table 6.4 Several Subjects’ Answers to the Questions in the Unit 8- Task 2 Worked Example
 Note: In the table, “not know” is used instead of “I do not know the answer” and Qn means Question number, Sn means Subject number.

The answers for Q3 (What would be a good name for this function?) are interesting, because there appear to be 5 different answers. However, “sum”, “total” and “sumOf” are different words for the same concept. At the moment, the raw data will be provided to the teacher. Analysing these answers cannot be done automatically, as it requires the teacher to make judgments. If asking multi-choice questions was supported with students selecting answers, it would make the automatic analysis simpler.

The answer for Q4 (What, if anything, should now come after the name sumList?) should have been “(theList)” which suggests that most of these students either misunderstood the question or did not understand the grammar requirement for defining a function in the Python programming language. The lecturer would need to decide whether to modify the question or change their teaching.

To some extent, by using the question responses in the log files the teacher is able to know how well students understand the contents, which were taught in the lecture. It is a passive method of collecting feedback from students. Although this method does not highlight precisely the problems the students had

with the materials, the correctness of the answers can provide some useful information to the teacher or worked example author.

6.4.3.2 Summary of the Findings from Log files

Based on the logging results, some key issues were identified and required further investigation. The questions needing further investigation are listed below:

- Why did so many subjects give “I do not know the answer” to the questions? Did they think about these questions before giving answers?
- The auto play mode was used more than expected, was there enough time for subjects to absorb all the contents in a fixed time?
- Why did so few subjects use the feedback function?

In order to get more information about some of these questions, some related questions were designed and asked in an online questionnaire. It is also shown that the question and answer mechanism can provide useful feedback to a teacher, but this function needs to be improved. Hence, an observation based evaluation is worth carrying out to deeply understand these questions.

6.4.3.3 Online Questionnaire Results

Based on the logging results, an online questionnaire was designed for subjects to evaluate their experience using IWE. The questions were designed to cover three aspects: general questions of usage and questions to identify why subjects did not use the feedback functions. Details of each question can be found in the Appendix 2. The results from these three aspects are described below and the raw data of these results can be found in Appendix 3.

General Questions of Usage

Subjects used IWE not only in the lab, but also used it in their own time. 26 subjects (65%) used IWE only in the lab, 7 subjects (17.5) used it only on their own machine and 7 subjects (17.5%) used it on both. The majority of subjects had no difficulties of using IWE. Only 4 subjects (10%) experienced difficulties caused by the Java Running Environment or to control the viewing. This result

showed that IWE did achieve one of the design purposes, which was to be able to be used flexibly.

22 subjects (55%) believed the tool was easy or very easy to use and 14 subjects (35%) thought the tool was neutral to use. The majority of subjects could use the tool to view the worked examples without difficulties. Problems were also found, based on other questions and subjects' open suggestions. For example, 20 subjects (50%) found the correspondence function and noticed the related contents could be highlighted automatically. Among those 20 subjects, 19 of them (95%) believed this feature were useful or very useful. The rate of finding the correspondence function was a little lower than expected, but subjects, who found and used this function, identified the usefulness of the correspondence function for interactive learning.

14 subjects gave very positive estimate of IWE and suggested some quite useful improvements. For example, a subject, who only used the examples in CS1P, said "the Pattern example helped me better understand the use of patterns, which I did not understand during the lecture, so quite a useful example and tool." This is exactly the educational purpose of IWE. 2 subjects suggested "designing a better example for demonstrating the functions of the tool" and they also requested more of this kind of example for the programming course (CS1P). Another subject claimed that "more of this kind examples plus proactive guesses would be more useful for their understanding of the subject." By providing some decision-making questions and the answers to explain the reason for making a decision, this request can be satisfied. Another subject said "the lecturer's explanations were easier to read while watching the example under self-control." Another subject mentioned that "the worked examples in IWE were better than PowerPoint slides or textbook examples as you can see how problems were solved in an interactive way."

Subjects also commented on the user interface. For example, a subject mentioned the "Your Answer" box looked like an edit box, which caused him to keep trying to type his answer in it. It should be more clearly marked as a read only widget. The step-by-step "Next" mode was preferable to the auto play mode. Another subject suggested using different font style or size in the explanation area, which could make the lecturer's explanations become more

readable at first sight. Another subject said “The interface can be improved; I did not like that example for the lab exam practice. I had to resize one of the windows at the middle. All other features seem to be OK. However, I prefer face-to-face talk with the lecturer if possible. If I do not understand something or the lecture notes, I can ask directly after the lecture.”

One subject claimed “he did not have any problems with the courses, so did not need this tool very much.” This suggested that the tool was an extra support for students who did not learn well inside the lectures. This is acceptable, as the tool was designed to offer extra support for students’ self-learning. Not every student needs it.

25 subjects (63%) believed using IWE could enhance their understanding of what they learned in the lecture. 13 subjects (32%) were not sure whether IWE could benefit their learning and 2 subjects (5%) thought the IWE did not work for them.

21 subjects (53%) believed the worked examples provided by IWE were easy to follow compared with worked examples in a textbook or reading through PowerPoint slides when working on their own. However, 17 subjects (42%) believed using IWE to learn on their own was the same as reading a textbook or PowerPoint slides. Some log file findings could indicate the reason, for example, worked examples were viewed in auto play mode occupied 55% and 33% total subjects gave “I do not know the answer” to every questions they met. This means some subjects did not use the worked examples in the expected way. They only passively watched the worked examples without active thinking and interacting.

These results indicated the worked examples in IWE were educationally valuable and these well-designed worked examples can be used for students’ self-learning. However, it is worth thinking about how to strengthen the motivation for responding to questions and make students feel more involved while they are learning on their own. It is also worth thinking about the play mode for students to control the viewing of the worked examples. As described before, the sample students were well distributed; it is quite hard to build a single worked example, which could be suitable for different level students. Hence, it is very important

to consider where and how to use the worked examples in IWE, which could maximize its educational value to most students. More than half of the subjects believed IWE worked for them and felt this new way of representing worked examples was better compared to the traditional way, which indicates the research is on the right track.

Feedback Questions

2 subjects (5%) sent feedback messages to the teacher after viewing the worked examples. A further multiple-choice question was asked to the remaining 38 subjects (95%), who did not send any feedback message. This multi-choice question is to identify the reasons stopping users from sending feedback messages. Because subjects were allowed to select more than one option as their reason, if a subject chose more than one reason then the selection was split between the options. For example, if two reasons were chosen then each reason was allocated a half vote, three reasons a third of a vote, etc. The most popular reason was that the worked examples explained everything very clearly, so there was no need to send feedback. About 21 subjects (55%) selected this reason. The other two popular reasons were “It took time to type in these messages”, which was selected by about 7 subjects (18%) and “I still preferred face-to-face talking rather than using this method”, which was selected by about another 7 subjects (18%). The other 3 reasons were very rarely selected.

Very few subjects sent feedback messages, but 35 subjects (88%) did not believe the feedback function was useless. 18 subjects (45%) believed it was useful. However, how to properly integrate this function into the tool was worth further investigation. Another question’s result supported this conclusion, which was that 30 subjects (75%) were confused by the feedback mechanism; one possible reason could be they did not know whether to write questions about the worked example or comments about the tool.

These questions’ results identified that a feedback function was a must-have function for IWE, however, to implement this function, especially the method to collect feedback messages, required more investigation.

Summary Findings from Online Questionnaire

The online questionnaire results show the IWE can be used flexibly and the delivered worked examples are usable. Some students also reported their benefits of using IWE, which pointed to the education value of the tool. All the functions designed for interacting with worked examples are necessary. The way of demonstrating worked examples is welcome; especially the embedded explanations for each individual step are very useful for the self-learning. The correspondence function is useful for students to build up the ability of making self-explanation, because of helping identifying the relation between the question and the solution. The feedback function is desirable, but the implementation of collecting feedback messages is not good enough.

The results did find possible answers for some issues, which were identified from the log file findings. For example, why did very few subjects use feedback functions? This could be caused by the implementation of providing feedback messages or the worked examples were designed and implemented well, so the students did not need to give feedback.

The result was also identified the reasons of why so many subjects gave “I do not know the answer” to the questions and did they think about these questions before giving answers? This could be because of requiring input of a lot of text to answer a question. Hence, ways of encouraging answering questions needs to consider, for example, reducing the amount of text input, or using multi-choice questions instead.

However, there are still no clues about why the auto play mode was used more than expected. For example, was there enough time for subjects to absorb all the contents in a fixed time? Can subjects find all the changes of a step in different panels and look at these changes at one time? In order to figure out the answer to all these questions, a one-to-one usability investigation could be helpful.

6.5 HCI Expert Evaluation

Philip Gray, who is an independent HCI expert, was invited to test the tool by running the *CinemaExample* without being given any training. The reason for

involving him was to collect some key issues from an HCI expert point of view, and then use them as guidance to improve the design of usability evaluation. An interview was conducted directly after he tried out the example and some useful and critical suggestions were given.

6.5.1 Suggestions Based on the Use of the Viewing Controls

1. The play button needs to be thought about more carefully. Watching the demonstration in auto play mode does not give sufficient time for the students to absorb all the information in the different panels, particularly, understanding the explanations. It may be better switched off at first and only made available if the user wants to use it.
2. While using the tool to step through a worked example, the user should be able to jump back to a particular step instead of clicking the “Previous” button several times. This requires adding an additional navigation method.
3. Highlighting problem: the highlighting should be exactly as same whether going forward or back.

6.5.2 Suggestions Related to Ask Questions

1. When asking the user to input the answer, the popup dialog box could confuse the user about he is doing, because the question is in the explanation view at the bottom of the GUI. It is better to add the question into the popup dialog box as well.
2. When the answer popup dialog box appears, it obscures the content of the worked example. In this case, the user might forget the context of the question. The system should allow the user to see the relevant content text.
3. When using manual control, stepping on should automatically jump to the next step after response to a question, rather than showing the question again.
4. When the student proceeds after answering a question, their answer should be shown together with the suggested answer in the explanation view. This allows the student to compare their answer with the model answer, strengthening the motivation to answer the question.

6.5.3 Suggestions for Future Improvement Work

1. Provide a facility for incorporating multi-choice questions. In this case, the user's operation is simply clicking on an option after they have thought about the question, rather than typing in text.
2. After exploring the *CinemaExample*, the HCI expert recommended that there should be a scenario to demonstrate the use of IWE to new users. He was shown the existing Warm Up demonstration but after he watched it, he suggested it was not good enough to train the user.

6.6 Follow-up One-to-one Usability Investigation

6.6.1 Overview

Some problems were identified in the previous survey that could not be explained through analysing the logging and questionnaire results given in section 6.4. So this usability investigation was conducted under the condition of one-to-one and side-by-side observation, which could offer the opportunity for the researcher to closely looking at how IWE was actually used. The scope of selecting subjects is more general in order to identify the problems that may happen during the usage, but the students who used the tool previously were not considered. Subjects were invited through email as volunteers to evaluate IWE.

Due to the ethics requirement, subjects were requested to read and sign a participation consent form before they took part in the evaluation. It was also required to ask subject's permission to use a camcorder to monitor their operations. Then the subjects could move on to the operating stage and were asked to finish tasks with minimum guidance; if they could not achieve the goal, or did not achieve the goal as expected, instructional directions could be given. During this operating stage, subjects were encouraged to ask questions or make comments about the functions or the operations needed to achieve a task. The observer stopped subjects at some special points to ask predesigned questions. Finally, an evaluation questionnaire shown in Appendix 4 was asked to be filled in. If subjects' time permitted, a short interview was conducted.

There were 20 participants in total, 7 tutors of level-1 courses, 5 MSc Students and 8 Level-1 students who had not used the tool before. All of the subjects

were familiar with the topic of the selected worked examples. The first 2 subjects who were Level-1 tutors were the pilot testers for this usability investigation; however, they provided high quality suggestions from both usability and educational aspects. So it was decided that these two results would be included in the usability investigation data.

6.6.2 Usability Investigation Questions

During the investigation process, at some special points the subjects were interrupted to respond to the observer's questions. These specific questions were designed based on: the results of the initial feasibility test; the feedback from Level-1 students' survey; and the independent HCI expert's suggestions. The reasons of asking them are explained below:

1. Students asked that the worked examples should be more interactive from initial feasibility test.
2. Highlighting colour caused confusion, which was gathered from open questions in the on-line questionnaire.
3. Help function is a standard function for any software tools. Hence, from a system usability point of view, it should be able to provide assistance while using the system.
4. Most students were not aware of the feedback function of IWE. Although they gave their reasons for not sending feedback messages, there may be other reasons stopping the use of this function.
5. Due to the content changes happening in several different panels, from the educational point, it may cause cognitive overload. Hence, it is required to ensure that subjects can notice all these changes without causing cognitive overload. Some questions, which related to subjects eye tracking, should be considered.
6. Comment on the "Play" button needs to be thought about more carefully, which is the first issue of the HCI expert. Choosing which mode to play was identified by the pilot testers for this usability investigation.
7. When asking a question, the popup box obscures part of the context. This is the fifth issue from the HCI expert and was also found in the pilot test of this usability investigation.

8. Whether the subject can see that the answer is recorded in the answer view initially is another issue from both the HCI expert and pilot testers. They suggested integrating the student's answer and model answer together in a view, could help students to compare these two models, while going through the examples. Hence, questions related to comparing answers should be considered.

6.6.3 Usability Investigation Process

The whole process of this investigation was explained to subjects first, after sitting in front of the screen. They started from the main GUI of IWE, so were ready to explore a worked example. *CinemaExPlay* and *FromERToTables* were selected as instances to carry out the testing tasks. The instruction details and the subjects' directions are described below:

1. Explain the whole process of this survey, which takes about 5 minutes. This includes:
 - 1) Tell them the objective of this evaluation, which focuses on the usability of the tool, not the content of the example;
 - 2) Generally explain the educational purpose of the IWE;
 - 3) Let subjects know that during the usage they should try their best to do things before asking me for extra support; but if they are stuck they can ask for help from me. They are encouraged to comment about the operations at any time and I may stop them to ask some questions as well.
2. Test the different functions of the tool taking about 20 - 25 minutes. This includes:
 - 1) Make the screen full size, which could more working space for watching and interacting.
 - 2) Ask them to the Run *WarmUp Demo* example and observe how they manage navigation for a while. Because there are two types of navigation, auto-play mode and manual control mode; at the end of this example the other unused navigation method needs to be pointed out to the subjects. If they press the 'Play' button and get lost then I should restart the example for them and suggest to them that they use the stepping forwards and backwards buttons, which switches to the

manual control mode, and explain the difference between these two control modes.

- 3) After completing this demo example, try to guide or lead subjects who cannot find the correspondence function. For example, by asking them “Would you like to interact with items in the documents or would you like to click on the graphical components in the graphical document?” This could make sure everyone is aware of correspondence before moving on to the main testing example. A question like “Do you think the correspondence function is useful?” should be asked as well.
- 4) Ask subjects whether they can see the change in the answer view after responding to a question. This question could be asked at different times, for example, just after finishing an answer or at the end of the example.
- 5) Give a brief summary to the subjects to summarise the major functions that can be used to look at a worked example.
- 6) Ask the subjects to run the main testing example, MSc students test the *RequirementToER* example, Level-1 students test the *CinemaExample*, and tutors test a particular example as appropriate. Before they select the example, they are asked to take the example as seriously as if they are working in a learning scenario. During this bigger example, subjects may be stopped at some point to be asked some questions. This could avoid subjects clicking buttons straight away without looking at the highlighting contents or changes in different views. I am always responsive to questions from subjects. The stopping point is selected at highlighting several components together. For example, in the *CinemaExample*, it could happen after highlighting a part of problem description document and a part of plan document, or a part of plan document and some part of the code document. In the *RequirementToER* example, it could happen after highlighting a component in the ER diagram document and some text in the requirement document. The questions to be asked include:
 - i. Do you look at changes in all three parts (document panels plus explanation panel) in a step or is there too much information for one step? Because it is necessary to know whether this way of

- presenting worked examples could increase students' Extraneous Cognitive Load.
- ii. What is your priority focusing; what do you look at first, explanations or changes in the documents panels? Because there are three representations (yellow highlighting component, orange highlighting component and the explanation texts) that need to be looked at in one step.
- 7) At various points, usually after subjects respond to a question, ask them whether they compare their answer with the model answer.
 - 8) Ask subjects to use help functions, if they did not use it during the previous test process.
 - 9) Ask subjects to look at the right bottom panel of the main GUI. Ask them to guess the purpose of this area. I will explain it if they do not guess it right and then let them try it out. This is to assess the usefulness of the feedback mechanism.
3. Ask subjects to fill in a questionnaire. Conduct a short interview about the tool, if time permits. This could take about 5 -10 minutes. This could provide an extra opportunity for gathering different level users' opinions of how to use IWE properly.

6.6.4 Results of Usability Investigation

The results of questionnaire and the observation show that no major problems can cause users to fail using IWE. 18 subjects (90%) believed they finished all the requested tasks. 16 subjects (80%) believed they could use this tool to run all the functions without any help. 19 subjects (95%) believed they could run all the functions by using the system help provided. The results also show that the user interface is user-friendly. 19 subjects (95%) selected the option "moderately user-friendly" or better. 12 subjects (60%) thought it very user-friendly. All the subjects thought it was easy to explore a particular worked example. Details of these results can be found in Appendix 5 and 6.

However, based on the researcher's observation and suggestions from interviews, several problems which can cause user confusion and affect using were found. The major findings can be divided into three aspects and summarised below.

6.6.4.1 Student User Interface Results and Suggestions

The student user interface is constructed of several views: scenario view, worked example view, explanation view, answer view and feedback view.

Scenario View did not give any trouble to the subjects. Through simple clicking on the name of the worked example in this view, they can begin to explore the selected the worked example. 15 subjects minimized this view in order to give more space for the worked example view. 2 subjects also suggested “it would be better if the scenario view could be minimized automatically when viewing the worked example.”

Worked Example View was usable to represent the worked examples. 17 subjects did look at all the changes happened in a step in different panels. However, there were some problems when subjects responding to a question. Details listed below:

- The answer popup dialog box obscures relevant context. When there was a question asked, the answer popup dialog was shown in the middle of the screen, which was on top of the worked example view. It would not disappear until some input was entered. It can be dragged to the side of the screen, allowing contents of the worked example to be seen most of the time. However, if contents contained too much information to be held in the panels, the subject could not scroll up and down or left to right to see the rest of the contents. In this situation, some relevant contents were hidden due to the panel size limitation. If the subject did not remember all the hidden contents, they had to give up answering this question. Based on observations, 6 subjects (2 tutors, 2 MSc and 2 Level-1 students) encountered this problem. 2 occurred when testing the *CinemaExample* and 4 occurred when testing the *RequirementToER* example. One reason that caused this problem was the speed of viewing. For example, when running the *RequirementToER* example, if using auto mode play, the waiting time was too short to see all the problem description texts; if using manual control mode play, the user pressed the “Next” button too quickly and was not able to remember all the problem description text. Hence, after the answer popup dialog box appeared, it

should still allow the subject to see all the relevant contents as needed, which means it should still allow the subject to drag the scroll bar of the panels in the worked example view.

- In the popup dialog box, the text area for writing an answer was too limited. 2 subjects (tutors) pointed this out, after writing answers. 3 subjects (1 tutor, 1 MSc and 1 Level-1 students) spent several seconds on finding questions, before they answered them. They suggested the questions should not be in the explanation view; questions should be integrated into the popup dialog.
- Comparing self-answer with the model answer when viewing is in progress. After answering the question, in the next step the model answer was shown in the explanation view and the subject's self-answer was recorded in the answer view. The subject may compare the self-answer with the model answer. This possible comparison was confirmed by the results of observation and an interview question. 16 subjects (80%) compared their answers with the model answers very carefully. 3 subjects did not compare them. Comparing self-answers with model answers is educationally valuable, as this can foster the self-explanation to achieve the learning gains. However, 5 subjects (1 tutor, 1 MSc and 3 level-1 students) suggested providing a standard view to show both the user's answer and model answer together, which can make the comparison much easier because their eyes do not need to jump between the explanation and answer views. 1 subject (MSc student) said "sometimes the answer was too long to type in and the model answer was shown in the next step, so she just needed to keep her answer in mind and compared it with the model answer instead of comparing two different views side by side."

Explanation View was recognized so important for this type of worked example. The educational value of providing explanations to each individual step was confirmed. 3 subjects (1 tutor, 1 MSc and 1 level-1 student) also suggested different colours and fonts should be used to the texts in the explanation view in order to emphasize the key words of the explanation.

Answer View achieved its design target, which was to record user's answer and make users aware of their answers, as 14 subjects (70%) noticed the change that happened in the answer view and 10 noticed the change occurred immediately,

after responding to a question. Considering this result together with the result of comparing a self-answer with the model answer, two aspects need to be considered carefully.

- Make sure subjects are aware of their answers and believe it is worth answering the question properly. As the HCI expert suggested, this could strengthen the students' motivation to answer the question.
- Make the comparison between the two answers easier, as subjects and the HCI expert suggested.

Feedback View did not work as expected; further research work was needed to improve it. An intervention aiming to find the reason why the feedback function did not work as expected was conducted. The researcher firstly asked subjects to guess the purpose of the feedback view. If they guessed correctly, let them to try it out; otherwise, explain the purpose and then let them test it. 8 subjects (40%) guessed the purpose of the feedback view correctly; and 12 subjects (60%) were not sure about it. Reasons causing confusion or failure included:

- They were not sure whether this function was designed for the lecturer or the developer; they did not know who would respond to their messages and how they could receive the response. 7 subjects (4 tutors, and 3 Level-1 students) had no idea about the purpose of this view and how to use it.
- Based on observations, the sequence of actions to send feedback messages was another problem. There were 3 subjects who pressed the "Send Feedback" button first, instead of writing questions or comments in the feedback view first.
- Some suggestions from subjects could also be considered as reasons for failure. For example, the word "Feedback" confused them; the text area was too small to write in; the index number in the view caused them to think that, when writing a message, it needed to link to a step number in the explanation.
- Another possible reason, mentioned by a tutor: "I thought the amount of the questions in *RequirementToER* was OK and covered most of the questions asked during the tutorial and lab." This confirmed the result of the previous survey where 66% of students chose the reason "The worked

examples explained everything very clearly, I understood everything.” for not sending any feedback messages.

Based on observations of using the interface to explore worked examples and interview results, it can be concluded that the subjects were satisfied with it. 7 subjects (3 tutors, 3 MSc and 1 Level-1 students) realized this main user interface was quite similar to the Eclipse IDE main user interface, or Microsoft Visio Studio main user interface. 18 subjects (90%) believed they finished all the requested tasks successfully, 19 subjects (95%) believed the user interface was moderately user friendly or better. 20 subjects (100%) believed it was easy or very easy to pick and view a particular worked example. 16 subjects (80%) believed they can use the tool properly without any system help and 19 subjects (95%) believed they can use the tool even better with the system help provided. However, problems caused subjects failed to use feedback function were identified, and the need to modify the answer view with an easy comparison mechanism was highlighted.

6.6.4.2 General Feedback about the Functions of IWE

Correspondence Function was actively found and used by 9 subjects (45%) (3 tutors, 3 MSc students, 3 Level-1 students). 3 subjects (15%) (1 tutor and 2 Level-1 students) found and used it after tips were given by the observer. 8 subjects (40%) (3 tutors, 2 MSc students and 3 Level-1 students) could not find it until told. This observation result did not match the result of question 8 in the questionnaire very well; as 80% of subjects believed it was easy or very easy to find the correspondence function, however, from the observation it is only 60%. It could be that after using correspondence function, some subjects changed their mind.

Help Function was actively found and used by 20 subjects (100%). There was 1 subject who was confused about the address bar on the top of help window, because she thought it was a search area for searching help topics. This confusion suggested a new function for further development of the help system, which is supporting searching topics, because nowadays users are already familiar with using a search engine when facing a problem. The result of question 20 also matches the observation result. 19 (95%) subjects believed the

system help was easy to use, which indicated the help function achieved its design purpose.

The control of exploring worked examples needs to be reconsidered and improved, which matched the HCI expert consideration of using the auto play mode more carefully. 50% subjects used the auto play mode when exploring the warm up demo example. However, when they began to explore the main worked example in a real learning scenario, 80% used the manual control mode or used auto play mode for only one or two steps then switched. 2 subjects used the “Pause” button to control the auto play, which was quite similar to the manual control mode, because the time for demonstrating each step could be controlled. If considering these 2 subjects as manual control users, almost 90% subjects used the manual control mode to explore the worked examples. The significant increase in use of manual control showed that when learning with worked examples, students prefer to work at their own pace. This also matched the result in the survey of learning needs, which was that subjects preferred a teaching tool they could use at their own pace.

The result of question 5 in the evaluation questionnaire also showed that no subject preferred to use auto mode. Only 3 subjects (15%) would like to combine the use of these two modes and 4 subjects (20%) would like to use auto play mode for reviewing after learning by using manual control mode. Several subjects explained that the time of understanding each step was not linear, so using manual control mode was more convenient.

Another useful suggestion for improving the control of exploring worked examples was that keeping the history of viewing, and allowing jumping to a selected step by a simple clicking, which was suggested by 2 tutors and 2 Level-1 students. For example, allowing a jump to a particular step directly, after this step had been seen, instead of clicking “Next” or “Previous” buttons several times.

Investigation of Extraneous Cognitive Load

A single step of a worked example in IWE is presented with several pieces of information together, which are shown in different panels. From the CLT point

of view, inappropriate presentation of the learning materials can increase Extraneous Cognitive Load, and accordingly affect learning results. Hence, it was necessary to investigate how students would deal with this situation. The intervention was conducted by the observer asking whether subjects were aware of these highlighted pieces, and how subjects were actually looking at them at some special points. These points were selected at various times, but followed one rule, which is that at least two highlights occurred in different panels in a step. This is because in these steps subjects have to observe at least three different things. For example, in a step of the “*CinemaExample*”, in the **problem description document panel** some pieces of text are highlighted in yellow and in the **plan panel** some text has been added which is highlighted in orange and the content of the **explanation view** is updated as well. The purpose of this step is to explain the reason why this highlighted piece of the plan is related to the highlighted text in the problem description.

The investigation results showed that subjects did look at all the highlighted pieces occurring in the selected step. They all have their own preference to look at highlighted pieces in different panels under different situations. Nine subjects, who selected the “*RequirementToER*” example, read the explanations first and then looked at highlighted contents in other panels. When looking at the highlighted contents, they always looked at the content that was not in their mind. For example, if highlighting occurred in the requirement and the ER diagram documents, they looked at the new highlighting parts in the ER diagram first; because the requirement was already displayed in the panel and remembered in their mind as well.

Because the learning process is under subjects’ control, no subject reported having trouble in looking at the contents in any step of the example. In fact, subjects only processed one piece of highlighted content at one time. They looked at the changes in a step one by one, and believed highlighting could help them to focus on problem statements and the useful part of the solution. This fits the highlighting design purpose quite well, which is reminding or revealing the internal relationship between problem and solution.

6.6.4.3 Tutors' Suggestions from Educational Points

The biggest advantage of inviting tutors as subjects is that from experience they know the real problems that occur when students are solving problems.

Therefore, they can suggest how IWE could be better used to support students learning based on their tutoring experience. Some good suggestions were gathered from the other two types of subjects, but tutors' suggestions are more educationally valuable. Based on the results of interviewing these front-end educators, some really useful suggestions are summarized into three major aspects.

Suggestions on How to Use the Tool

- 5 tutors and 1 MSc student suggested introducing how to use the different functions of the tool in the classroom. For example, explain the correspondence and the feedback function, as they are useful while students are self-learning. They are easy to use after being introduced. They also suggested demonstrating the examples in a lecture was necessary, so students could know these examples were designed by the teacher to support their learning.
- 1 tutor suggested it was better to emphasize using the feedback function at the last step of an example. Tips or other methods can be used to remind students to send questions or comments, if they have any.
- 1 tutor suggested it was better to only allow using manual control mode for the first time of viewing the example in the real learning scenario situation. If students would like to review the example again, then the auto play mode could become available to use as it is more efficient and convenient for reviewing.
- 4 subjects (3 tutors and 1 Level-1 student) believed that if these examples can be delivered earlier to the CS1Q course, it definitely could help the students understanding of the whole transform process of designing an ER diagram based on a textual description. They believed if these worked examples can be well integrated with the courses then more students could benefit from using them.

Suggestions on When and Where to Use the Tool

- 2 tutors suggested it was better for students to use IWE to see these examples in their spare time before they dropped into the lab. If they did, some common questions asked during doing exercises would be avoided. For example, how to deal with the “Multi-Value” attribute when transferring ER to tables. From this point of view, it shows that using IWE could reduce the tutors’ workload in the lab. 3 tutors confirmed this, as they explained there is no need to repeatedly answer the same question again and again to each individual student. A Level-1 student after exploring the *ERtoTables* example, who mentioned he definitely learnt something, which did not understand well before, confirmed this. He stated that “if I saw this example before dropping into the lab, I would not ask how to deal with a “Multi-Value” attribute in an ER diagram.”
- 1 tutor commented that the IWE could also be used in the lab. Because this kind of worked example provided a starting point of how to apply previous knowledge into problem solving, it could support students to build abstract thinking processes. By using these kinds of worked examples, a tutor can answer students’ questions better because sometimes some concepts are hard to explain in spoken language. After she saw the Programming Pattern example, she confirmed her idea. She commented that this kind of worked examples showed students how to apply their previous knowledge. It was not only focusing on introducing the concepts, but also demonstrating the procedure of solving problem, which was the biggest challenge Level-1 students struggle with.

Suggestions on Enhancing the Instructional Design

- 2 tutors suggested adding more explanations for “Multi-Value” attributes in the DB relations example.
- 2 tutors suggested it was better to give more explanations about adding attributes to a relationship in *RequirementToER* example.
- 2 tutors believed that the correspondence function was extremely useful for level-1 students, especially, useful for doing the transformation from ER design to tables. Because foreign keys were missing in the ER designs, he had to explain again and again to each individual student how to add foreign keys into tables.

- 1 tutor and 2 level-1 students suggested if possible, it was better to set up the correspondences in *CinemaExample*.
- Another Level-1 student wanted some extra explanations of step 40 in *CinemaExample*, which could explain how the “print msg” can be linked or mapped with the plan.

These suggestions include the expected types of feedback, which should be received by the teacher so that they can use these to evolve the worked examples, where appropriate.

6.7 Conclusions

This chapter illustrated the first 7 evaluation results, which were conducted in two stages of IWE development. The initial prototype evaluation results identified the need for comprehensive interactive worked examples with explicit teacher’s explanations for novice students. So the teacher should be able to create these kinds of worked examples to meet the students’ requirements.

The teacher’s evaluation results showed that it was possible to use IWE to build interactive worked example in a reasonable time without doing any bespoke programming. A single teacher could build a worked example in a reasonable time to fit his or her own teaching style. A number of different styles of worked examples - Graphical to Text, Text to Graphical, Text to Text, and Text to Text to Text, were created and delivered to the students to use during the normal teaching process. The teachers realized the value of using IWE in their teaching.

However, to use IWE to fit in with their future teaching requires the researcher to do some further development in order to meet their expectations. For example, improving the teacher user interface to include: making the creation process much more intuitive; adding extra formatting mechanisms for editing textual documents and the explanatory texts; making updating textual documents more flexible after being used for defining a process; adding a grouping mechanism and operations on a set of fragments when defining a process. Little significant feedback was collected from students but Log file results can be used as a kind of students’ feedback. Teachers’ self-reflection, while checking and reviewing of worked examples before delivering to students

can also contribute to evolving the worked examples, but this kind of evolution was not expected. IWE needs to implement a better feedback mechanism, so that evolving worked examples can be based on students' feedback. Although the creation of worked examples required a lot of intellectual effort, teachers found past exam questions and examples used in lectures and tutorials can be good resources to be used for IWE.

The level-1 students' survey and final usability investigation results showed that the overall subjects' experience of using IWE was that 95% of subjects could use all the functions of the tool and believed the user interface of IWE was user friendly.

The usability investigation results indicated that representing worked examples in IWE did not increase Extraneous Cognitive Load. The worked examples delivered could promote subjects to actively think about a problem, and then understand the teacher's strategy of applying knowledge to solve the problem, which is supported by applying the highlighting technique. Subjects did like these kinds of examples and suggested creating more for their learning. So, these worked examples are usable.

The implementation of the feedback function needs to be reconsidered and redesigned. Some subjects did not actively respond to the embedded questions or provide reasonable answers. Questions, like how to encourage students to give feedback that will assist teachers in evolving worked examples and how to encourage students to answer the embedded questions so that their answers can be provided as feedback to the teacher are worth further research and investigation. Some good suggestions about the use of IWE given by the students and tutors can be taken into account and these are discussed below:

- Introduce or demonstrate IWE in the lecture, including the use of the correspondence and feedback functions. This could encourage students to use these functions more often and properly.
- Explain the benefit and value of answering questions to students as an alternative way of giving feedback to the teacher.
 - For students, they should immediately get the answers of the question, so they can compare their answers with the teacher's

model answer in order to check whether they were engaged with teacher's expectations.

- For teachers, students' answers can be used as a guidance to improve the teaching. For example, if so many students answered a particular question incorrectly, the teacher can explain it in a future lecture or tutorial, alternatively, the teacher can modify the worked example by editing the process steps or improving the explanations of the steps.
- Integrate the use of worked examples in IWE with tutorials and labs, which could benefit tutor's working. Integrating worked examples with exercises in the laboratories also follows best practice recommendations for using worked examples.

6.7.1 Enhancing the User Interfaces

Most of the issues with IWE identified belong to implementations of IWE, which can be improved through modifying the user interfaces. These improvements are necessary, as by modifying these problems, the Extraneous Cognitive Load for students can be further reduced to improve the usability of the tool. Suggested improvements for enhancing the user interfaces are discussed below.

6.7.1.1 Short-term Improvements to the Student User Interface

- The auto play mode should be removed from the process player. Students should only be offered the Step buttons, so that they work through the examples step by step in Manual Control Mode. Because the contents in each individual step are different, the reading speeds for each individual student are different; it is impossible to set up a standard fixed time for viewing different steps. Students who use the auto play mode passively receive information and so they do not immerse themselves into the worked examples. By using the manual control mode they are at least actively doing something and to some extent involving themselves with the worked examples.
- The way of controlling exploring worked example should be more flexible, for instance, allowing the user to jump to specific points which were viewed instead of clicking the back or forward button several times, or keeping the history of previous viewing steps allowing the user to

continue the previously interrupted viewing. A navigation bar could be added to replace the current progress bar. It can list all steps of the worked example and show the current step position and the last step position. Jumping between different steps, which have been explored, can be achieved by choosing the steps number within the navigation bar. Adding this facility will allow students to quickly access earlier steps as required instead of clicking the Previous Step button repeatedly. An equivalent of an Internet browser's "Back" and "Forward" buttons could be used to aid navigation.

- The question and answer mechanism needs to be improved, especially multiple-choice questions should be added. Using multiple-choice questions could be a good method of improving question and answer mechanism, as it only requires students to select one or more options. If the question is a descriptive question, the popup dialog box should contain the question text and a textual area for the student to type in the answer. If the question is a multi-choice question, the popup dialog box should contain the question text and the options that can be selected. The answer popup dialog box should never obscure relevant context, it should allow to be moved into other position on the screen. The actual answers should be recorded automatically, and saved on the server.
- When the student steps on after answering a question, the explanation view should display the student's answer; followed by the teacher's answer (Your answer was ... Here is the teacher's model answer ...). The student's answers should be recorded so that they can review them later, if necessary, and the teacher can see them. Meanwhile, by combining the student's answer with the teacher's model answer, the comparison becomes much easier, as the student's eyes only need to focus in the explanation pane. Using multi-choice question reduces the required amount of input from students and increases the motivation for answering a question. Explanations can be given about the incorrect answers and why they are wrong. Teachers will be able to obtain quantitative feedback quickly through analysis of correct and incorrect choices and make changes to the worked examples or make adjustments to future teaching activities.

- The user interface for writing feedback messages needs to be improved, as it was not obvious what it was for and how to use it. Although 40% of users from the follow-up one-to-one usability investigation knew the purpose of the feedback view, they misunderstood the sequence of using it and confused the index number for listing feedback messages with the step numbers. Hence, the feedback view should be removed from the student's main user interface and a 'Feedback' button added to the menu bar. A popup dialog box inviting students to write the feedback message will be shown if the 'Feedback' button is pressed. A feedback message will be sent away if the student clicks the 'Send Feedback' button in the popup dialog box and the box will disappear. Meanwhile, at the beginning of a worked example tell the students what feedback is (questions, comments, etc.), how it will be used (emphasise its value to the teacher and ultimately to students, in the form of improved examples) and that they can use this button as required. At the end of a session, invite the student to give any final feedback about this worked example.

6.7.1.2 Short-term Improvements to the Teacher User Interface

- A new user interface for teacher to design multi-choice questions needs to be provided during the process definition.
- IWE should be able to provide an automatic report that summarizes all the students' answers of multi-choice question to the teacher. This is a passive way of collecting students' feedback, but it is an effective way to gain insight into how well students are learning. It also allows the lecturer to identify common misunderstandings in answers to questions, which may suggest the need for changes in the worked example or other teaching.
- An import and export function should be added. It should allow a teacher to import or export a particular application. By adding this, some definition work can be sped up and it could also allow different teachers to collaborate with each other remotely to build worked examples together.

6.7.1.3 Long-term Improvements to the Teacher User Interface

- The user interface for the teacher to construct graphical documents is very basic and unfamiliar. It needs to provide a more familiar graphical

document building interface with a palette of shapes that can be dragged and dropped into the drawing panel (or a point and click interface). It should allow the teacher to define customized shapes as needed. A zoom in and out function could be another useful facility when constructing the graphical document.

- The user interface for the teacher to construct textual documents needs to be improved. The current method for formatting text is very basic and unfamiliar. A new simple word processor interface is required including a set of styles, which are defined textual fragment types, which can be applied to the text while identifying the textual fragments.
- The explanation view should allow the use of more fonts to format the explanation texts, or adding background colour to highlight some keywords. For example, supporting emphasis (bold, italic, underline), different text sizes and more than one font (fonts that match those used in related text documents would be useful). A simple text editor similar to those used to compose a basic e-mail message could be considered to replace the current explanation text input pane.
- The process editor interface needs to be modified. When defining a process, most actions involve choosing a fragment from a list of fragments for a given document and specifying the action required. It could be made easier by allowing the teacher to select fragments from a text document and apply appropriate operations (insert, highlight, unhighlight, delete). Adding 'Move' operation to graphical fragments, which allowing moving one graphical fragment from one point to another point, is desired, especially for creating worked examples, such as explaining the concept of heap and stack in a programming course.
- Currently, highlighting a number of fragments in one step is tedious, especially if they then have to be unhighlighted in a subsequent step. The highlighting mechanism in process construction would be more efficient if groups of fragments could be highlighted or unhighlighted in one command. Furthermore, allowing the teacher to define composite types of fragments would be an even better solution. Grouping several different types of fragments to become a composite fragment first, and then provide add, highlight and unhighlight operations for this composite fragment during the process construction. Composite fragments could also

be useful for defining the correspondence between multiple fragments in different documents, when teachers are defining the correspondence function.

- The flexibility of document modification after process definition needs to be improved. Currently, the major deficiency comes from the inability to be able to make changes to the structure of a document during process definition. It is possible to edit the content of an individual fragment but not to insert new fragments into the document. This problem stems from the choice of XML to represent the document structure. It is not possible to change the structure of a document part way through the definition of a process because this would change the reference numbers of the fragments and result in corrupted document. It would also mean that any processes sharing the current document would fail to run properly. The implication for the existing system is that all supporting documents have to be meticulously prepared before committing to developing a process and explanations. To overcome this problem, a major redesign of the process document structure would be required.

6.7.2 Final Evaluation of an Improved Version of IWE

Some problems of the student user interface were identified that can cause Extraneous Cognitive Load when students are using IWE. There were also desirable changes to the question and answer mechanism that impacted on both the student and teacher user interfaces. Hence, it was necessary to carry out a third development cycle to make short-term improvements and then evaluate IWE from the usability and utility levels as well as to investigate how to use feedback to evolve the worked examples. The evaluation also needed to fully integrate the use of worked examples with the teaching following the best practice suggestions from education research recommendations. Chapter 7 describes a field study that aimed to fully evaluate the use of the final prototype of IWE integrated with the teaching of a course for novice programmers.

Chapter 7 Field Study of IWE

The previous evaluation results have identified deficiencies in IWE, especially showing that the student user interface needs to be improved. Taking these results into account, the third development cycle, focused on improving the student user interface, was finished and a new version of IWE was released for conducting a field study. The purpose of this field study is not only to evaluate the usability of the tool, but also to investigate how useful IWE is in a real educational context. Such a context should have the following characteristics: the subjects are novices; the material to be learned includes problem solving; the current pedagogy includes lectures and lab-based exercises; class sizes are large enough that one-to-one tuition is infeasible. Several evaluation methods are used in order to demonstrate that IWE is usable and useful in the chosen context, described in section 7.1.1.

This field study overcame the major limitations of the last evaluation, which was a lack of demonstration of how the lecturer used the tool, and that the worked examples were not well integrated with the lectures and laboratories. This time the CS1CT course was selected and IWE was introduced in a lecture and all the worked examples for assisting students learning and doing exercises were well integrated with the teaching process and delivered at appropriate times. One set of worked examples followed the educational research recommendation that the best way of using worked examples is for students to look at a worked example first, and then do a similar exercise, repeating this cycle several times. This assisted students doing exercises in the lab. Two other sets of worked examples were delivered for students preparing for the lab exam and the written exam respectively.

Section 7.1 describes the educational context for carrying out this field study and the different educational purposes of delivered worked examples. Section 7.2 reports the results for the usage of IWE for all users and explains how to categorize the users into different groups in order to prepare for evaluating the usability of IWE. A serious user group was identified as the target group to evaluate IWE in both the usability level and utility level. The target group's levels of use of the examples are also reported. Section 7.3 reports the effective modifications, which were made based on the discussion in section 6.7.1.1 and

6.7.1.2. It also reports on the usability of IWE measured using SUS. Section 7.4 discusses the utility of IWE from a conceptual level and analyses the possible reasons for some subjects giving IWE a low score in the SUS Questionnaire. Section 7.5 uses two delivered worked examples as instances to report how the teacher can use feedback to evolve them. A possible way of further evolving these two selected examples is also addressed. Section 7.6 summarizes the major findings of this field study, and concludes that IWE was useful at both the usability and utility level. Lack of data about detailed measurement of IWE in supporting educational effectiveness is the major limitation at the moment; however, based on the current research results about using worked examples, the education value of the current version of IWE can be predicted.

The ethical considerations involved in this field study are similar to those of the classroom intervention described in 6.4 of Table 6.1 and the student observation of 6.6 in the same table. Students were not required to participate in any of the data collection activities, while still able to gain the full benefit of having IWE embedded in their classroom activities. All data reported is fully anonymised.

7.1 The IWE-based Interventions

This section starts by introducing the educational context for carrying out this field study. It then describes how IWE was used in an introductory programming course, the contents of the delivered worked examples and how they were actually used for different educational purposes.

7.1.1 Educational Context for Using IWE

The CS1CT course is designed for novice students, who have no previous computing education experience, to develop the necessary understanding and problem-solving skills in computational thinking. It contains 2 x 2-hour lecture/discussion sessions weekly Wednesday and Friday and 1 x 2-hour lab session scheduled flexibly from Monday to Wednesday. The lecturer started by introducing Alice[111], which is a graphical programming environment that can be used as a teaching tool to learn fundamental programming concepts. This was followed by teaching programming in the Python language. For the study, there were a total of 103 students enrolled in this course. On the basis of an early

questionnaire, the lecturer reported that approximately half of the students had some previous programming experience. All the worked examples delivered with IWE were designed to help learn Python, as programming in Python is harder compared with programming in Alice [111]. For novices there is a bigger conceptual gap between the problem statement and the executed code when using Python. IWE was designed for promoting students from novice level to intermediate, and so these CS1CT registered students were the best users for evaluating the tool.

7.1.2 The Use of IWE in CS1CT

The field study was carried out in the first semester of the 2013 to 2014 academic year, using the improved version of IWE. Over a period of five weeks, starting in week 7 of the course, 8 new worked examples were created by the CS1CT lecturer during the busy academic teaching period. They were delivered to the students, with the purpose of assisting students to better understand the concepts which were introduced in the lecture, or helping to finish exercises. They were provided in three stages, the first stage was planned in advance in the middle of the teaching programme, and the second and third stages were in response to a realisation that students weren't sufficiently prepared for two assessments at the end of the course.

Students could access examples either in the lab or by downloading them to their own computer. In the lab, there was a shortcut icon named "IWE" on the desktop screen after they logged in to use the computer. Students double click on the icon to use the IWE. A simple set of instructions describing how to use or download IWE was written at the beginning of the lab sheet. IWE was demonstrated in the Friday lecture immediately before the following week's labs and these instructions were for students who had missed the lecture or simply needed reminding how to access IWE.

The support for updating worked examples is different for students using IWE in and out of the lab. If a new worked example or an updated version of an existing worked example is released, the teacher only needs to publish it on the internal server. In the lab, students can use this worked example directly without any extra work. The activities of viewing worked examples are recorded in the log

files and automatically sent to the server, together with the students' answer files and feedback message files. However, out of the lab, due to the limitation of current deployment method used, if there is an update of a worked example or a new example is added, students download the whole IWE package. Log files could not be collected from remote users, but the answer files and feedback messages files were collected through using a default Gmail account.

7.1.3 Planned Use in Lab Exercises

The first batch of IWE examples, which aimed to support practical exercises, was released at 9 am on 11/11/2013. Details about the whole set of exercises can be found in Appendix 8.

The first example was called *DrawingClockProgram*, which demonstrated how to write a program to place the numbers 1 to 12 as if they were on a clock face. It used 18 steps. Students were directed to view this example first, and then tried to solve a related problem, which was to draw a bicycle wheel - a wheel rim and 8 spokes running from the centre of the wheel out to the rim. The exercise is very similar to the *DrawingClockProgram* in IWE, as the calculation for the end points of the lines for the spokes is the same as for the positions of the clock numbers. This is directly in line with the best practice recommendation for using worked examples.

Three further worked examples were provided, making use of a slightly different instructional cycle, recognising that this is a mixed ability class. Some of the students who had previous experience of programming on entry to the course were effectively intermediate learners and needed a challenging approach to learning. The novices, without previous programming experience, progressed at different speeds depending on how well they had understood the concepts of programming in Python and coped with the transition from Alice. They required more structured support.

In the modified learning cycle, students were asked to solve a new problem first. If they could not, they were invited to view a worked example in IWE for that new problem; if they could, after working out the solution to the problem, they were invited to review the worked example in IWE and to compare their solution

with the example solution. Then they were invited to solve another problem, related to the first.

The second exercise in the lab sheet starts with a problem to draw 10 parallel horizontal lines, each 100 long, starting at (10, 10) and spaced 10 apart, going downwards. The corresponding worked example in IWE was called *HorizontalLinesProgram*, which demonstrated how this problem was solved in 19 steps.

The related problem is to draw a grid of cells (like an empty table) with 5 cells across and 4 cells down. Each cell should be 30 wide and 20 tall. Students were asked to develop the solution for the related problem, as this problem can be solved by modifying the related IWE example and then drawing extra parallel vertical lines.

The third exercise in the lab sheet starts with a problem to draw a random segmented line, which should start at (100, 100) and consist of 10 straight segments, connected so that the end of the first line joins up with the start of the second, and so on, to form one long line. The length and orientation of each segment should be chosen at random, between the limits 0-50 and 0-359 respectively. The corresponding worked example in IWE was called *RandomLineProgram*, which demonstrated how this problem was solved in 16 steps.

The related problem in the lab exercise was to draw a regular 5-sided shape, a pentagon. These two problems are similar, as the difference is the angle between lines and the length of the lines, in the former these are random values and in the latter they are fixed values.

The fourth exercise in the lab sheet starts with a problem to adjust the student's own pentagon program to provide a function to draw a regular polygon of any size (number of sides and length of sides).

The corresponding example in IWE was called *PentagonToShapeProgram*, which demonstrated how to draw a regular polygon of any size in 17 steps. It focused on how to make use of a function that takes parameters to implement the code.

The related problem was to ask students to go back to the previous program to draw a grid (fixed number of cells and cell sizes) and turn this into a function, which can take parameters, to draw an arbitrary grid (number of cells across, number of cells down, cell width, cell height can all be varied.).

These three examples explain, step by step, the thinking process of how the teacher worked out the solution from the problem statement. Multiple-choice questions were also embedded to encourage students to think about the solution actively, when there is a decision to be made. Students' answers to these embedded questions were collected and provided to the lecturer as feedback at the end of day, after all the students finished their lab session.

Modifications to the worked examples in IWE were made, based on students' answers and feedback messages received. Details of modifications will be provided later in the teacher's evaluation section.

7.1.4 Deepening Conceptual Understanding

The lab exam question (in Appendix 9) was proposed to the students in 18/11/2013, two weeks before the lab examination. To solve the problem, it is required to read data into a list, process data from the list, use the list as a parameter to a function and output data from the list. As the teacher realised that the students may not have had sufficient preparation for writing functions using lists, he decided to make another worked example to help students to prepare for the lab exam by deepening understanding of this particular concept.

On 25/11/2013, a new example called *ListProcessingFunctionExample* was released. The question asked was to write two functions, which should both take a list of words and they should both check each word to see if the first letter is upper or lower case. For any word that has a lowercase first letter, the first letter should be converted to upper case. The first function should create and return a new list containing all the words, and the second function should change words in place in the list - it does not need to return anything. This worked example was designed to help students understand how to use functions to process lists; understand precisely how lists can be passed as parameters to

functions; and lead them to follow the thinking process involved in problem solving.

The way of demonstrating the solution in 19 steps was slightly different to the earlier worked examples. The first 7 steps form an intermediate step, which is used to create a high level plan. The remaining 12 steps are used to explain how the solution is generated by translating this plan into programming code. It also contains an embedded question in order to force students to think about the solution rather than passively receive it.

ListProcessingFunctionExample can help students to build up a way of thinking about how to work out a possible solution for their lab exams. It related to the lab exam question involving processing lists and writing functions. For example, the third requirement of lab exam question was to create a function to remove stop words from a list of words that had been read in. This required using a stopWords list as a parameter for coding the function, which checked the whether a word in a list of words read in was in the stopWord list. This was quite similar to the first function of *ListProcessingFunctionExample*, which is to create and return a new list containing all the words.

7.1.5 Practice with Exam Question Formats

On 16/12/2013, another 3 big examples, to assist students in preparing for the written exam to be held on 18/12/2013, were released. The 27th question in the written exam asked students to write a complete Python program which allows the user to type a string at the keyboard and then display in four columns how many times each letter of the alphabet occurs in the string. It is assumed that all letters entered will be in lower-case, and any other characters should be ignored. (Details of this question can be found in Appendix 10.) So the teacher created two new worked examples in IWE, similar to the written exam question, to help students practice and prepare for the written exam.

PasswordCheckerExample and *JustAMinuteExample* are all about analysing the contents of strings, which are problems related to the 27th exam question.

PasswordCheckerExample contains 31 steps, which asked students to write a program to validate a password from user input. The conditions are: it must have

at least 6 characters and at least one digit, one lower case letter and one upper case letter.

JustAMinuteExample is to simulate aspects of the Just A Minute Program on BBC Radio 4, which asks contestants to speak on a specific topic for one minute without repeating any words. So the solution of this example demonstrates how a computer program can help the game host to analyse the contestants' performance. It needs to read in a single line of text, representing whatever the contestant said, and write out any words that appear more than once in the text. The whole process of working out the solution was represented in 20 steps.

The 25th and 26th questions in the written exam asked students to find errors in the code implemented to solve a small problem, explain the reasons for the error and correct the code. Each of them contains 3 errors. (Details of these two find errors questions can be found in Appendix 11)

FindErrorExample was a related example to this type of exam question. It was designed to support students in applying strategies for identifying errors, giving an explanation of the identified error and correcting the incorrect code. This example is a special example, as 6 strategies for identifying errors in the code were well explained in the different steps. The errors can be found by applying one of these six strategies, which can be directly applied to answer this type of exam question. It contains 29 steps, which asked students to identify the errors in the programming code and then fix them. During the process of identifying errors in the original code, related questions were asked. These questions were either on making judgment about whether a line of code is right or not; or writing down the explanation of the identified error by applying the relevant strategy, as practice for doing the same in the exam itself.

These four examples are not directly following the best practice recommendation in the field of worked example research, due to the time period limitation between showing worked example and solving related problems. However, they can demonstrate how the problem could be solved by the lecturer and how the lecturer used taught knowledge to solve the target problem. As these four examples are bigger examples comparing with the first

batch of worked examples, these four examples aims to provide more experience to the students, which can help students to build up their own experience when they try to solve problem on their own. Based on the discussion in chapter 2, especially regarding Laurillard [32] and Kolb's [33] suggestions of building experience, the more worked examples students can see, the more experience they gain, which can be used to build better solutions. The way of using these four worked examples is still following the recommendation to show a worked example and then ask students to solve a related problem, suggested by Trafton and Reiser [40].

7.2 Users Identification for the IWE Evaluation

During use of IWE in class, 31 feedback messages (29 feedback messages from the server and 2 feedback messages from email), 186 answer files (102 answer files from the server and 84 answer files from email) and 116 log files were collected as a part of the data resource for preparing to test the usability of IWE. The log files statistics showed that these examples were used by 70 students in the lab. The answer files received by email showed that these examples were used by 40 students on their own computers. Based on these two figures and the total of 103 students enrolled on the course, it is shown that some of the students used IWE not only in the lab, but also out of the lab.

IWE is an education software tool, which will have high utility if students learn from using it. As discussed in the usability review section in chapter 3, in order to evaluate IWE in both usability and utility level it is necessary to identify suitable users from the whole class of students. From the CLT point of view, the Intrinsic Cognitive Load can only be provided at a suitable level for the majority of students, not all the students can benefit from using the worked examples delivered by IWE. Hence categorizing students into different groups is essential, based on their ways of using IWE. Students who had serious sessions with the tool should be the focus group, as they probably engaged with the worked examples, if the level of Intrinsic Cognitive Load is suitable to them. When evaluating the usability and the utility of IWE, their suggestions are more valuable comparing with users who rarely used IWE. There are three different data resources that can be used to categorize students, listed below.

1. IWE provided the function which allows students to send some feedback messages.
2. IWE also provided the function to record students' answers and the time cost for answering a particular question, if there were embedded questions in the worked examples.
3. IWE can record students' detailed activities of using the system into daily rolling log files, if they used IWE in the lab. For example, the name of the worked example, the time of starting viewing, the time spent on each individual step, and so on. The usage log file will be automatically sent to the server. These students' IWE log files can be used as supplementary data to make judgements on how students actually used IWE and how engaged they were. However, if students used IWE on their own computers, no log file can be sent to the server, due to the technology limitation of finding the log file path on their own computers. This problem could be solved by creating a web-based version of IWE for student use. This is discussed in section 8.3.1.

Based on these three different data resources, students can be divided into three different groups: serious user group, light user group and non-use group. The criteria for adding user into serious user group are described below:

- Students sent feedback messages, especially talking about the contents of the worked example, or asked questions about the example, or mentioned problems during the viewing process.
- Based on the time cost for answering a particular question, which can be found from the students' answer file, students spent a reasonable time answering the question. In contrast, students who simply clicked through avoiding the questions were not deemed to be using the tool seriously.
- The time cost for viewing each individual step of a worked example, the number of steps which were viewed and the time for viewing a particular worked example can be found from the log file. Serious students should spend a reasonable time on each individual step and view at least 80% of the contents of the worked example. Reasonable time was decided by the teacher as part of their analysis of expected usage for a worked example.

If students met any one of the conditions mentioned above, they were added into the serious user group. For these users, the proper usability evaluation method should be applied.

If students did send feedback messages, but did not talk about their learning, their answer files should be checked. If a very short time was spent on responding to questions, they were added into the light users group. If no feedback messages or answer files were found, the log file was checked. If students only viewed very few steps of a worked example, or viewed all the contents of a worked example but only spent a very short time on each individual step, they were treated as light users as well. For these users, finding out why they did not use IWE very often, or did not use it as expected, is the major task.

If no feedback messages, no answer files and no log files were found from either server or email, they were added into the non-use group. If no log files were found for a student then this means that they did not attempt to use the system in the laboratory. It is likely that these students were struggling with the course and had stopped attending laboratories, or they were sufficiently advanced that they did not feel they needed to use IWE. They might have used the system exclusively at home but the absence of any answer files sent by e-mail suggests they did not use any of the worked examples with embedded questions. These students were not considered for evaluation of the IWE, as they were probably not system users.

Finally, the 103 registered students were identified into 3 groups: 54 were serious users; 19 were light users; and 30 were non-users.

The serious user group is the focus group for evaluating IWE. Compared with the light user group, they spent more time on using the tool to assist their learning and had better understanding the value of the tool. Hence, their evaluation results are more valuable for the prototype of IWE. Different strategies should be applied to the two groups, for instance, serious user group can use a mature usability evaluation method, and the light user group only needs to be investigated to understand why IWE was not used in the expected way.

A detailed analysis of log files and answer files, related to the 54 serious users of IWE was carried out. Several typical usage traces from the log files will be presented later on. A part of this analysis is shown in Table 7.1, which shows the frequency with which each example was completely viewed by these serious users. The last 3 examples were used very few times in the lab; this was because the release time was only 2 days before the lab exam and very few students dropped into the lab. So only a few log files were recorded. However, the answer files received by email showed that students used IWE on their own computers and *FindErrorExample* was viewed by 27 users out of lab. Because the *PasswordCheckerExample* and *JustAMinuteExample* do not contain embedded questions, no accurate figures can be reported. However, as these three examples were released at the same time, the lab usage data shows that they were viewed a similar number of times by a similar number of users. It is possible that similar numbers of users viewed these other two examples out of the lab. Table 7.1 shows the number of each worked example that was viewed by users in and out lab. The number was counted based on the answer files received on the server and by email.

Example Name	Number of Serious Users in Lab	Number of Serious Users out of Lab	Average Viewing Time in Lab (Seconds)	Embedded Questions
<i>DrawingClockProgram</i>	29	N/A	912	No
<i>HorizontalLinesProgram</i>	25	14	602	Yes
<i>RandomLineProgram</i>	17	8	1026	Yes
<i>PentagonToShapeProgram</i>	9	4	879	Yes
<i>ListProcessingFunctionExample</i>	21	24	969	Yes
<i>PasswordCheckerExample</i>	4	N/A	769	No
<i>JustAMinuteExample</i>	5	N/A	810	No
<i>FindErrorExample</i>	4	27	821	Yes

Table 7.1 The Number of Serious Users in and out the lab, who Completed Viewing a Worked Example

Although the light user group was not the focus group, it is worth investigating why they did not use IWE very often. Hence, an open question was asked to this group in order to investigate some possible reasons. Finally, 11 light users answered this question.

7.3 Usability of IWE

7.3.1 Effectiveness of Short-term Improvements

Short-term Improvements, which were described in the section 6.7.1.1 and 6.7.1.2, were made. These modifications were shown to be effective in the new version of IWE, which are described below.

1. By modifying the user interface for writing feedback messages, in this field study, 31 feedback messages (29 feedback messages from the server and 2 feedback messages from email) about the worked examples were received. This number is more than the evaluation carried out a year ago, which only received 4 feedback messages.
2. Adding the multiple-choice questions and providing an automatic report that can summarize all the students' answers of multi-choice question to the teacher is also effective. There were 186 answer files (102 answer files from the server and 84 answer files from email) received. Not only has the number of answer files increased, but also the teacher can use the summarized report to improve the released worked examples. The details of the modifications made by the teacher will be discussed in section 7.5.

7.3.2 SUS Questionnaire Results

An SUS questionnaire was sent out to 54 serious users to be filled in by using the SurveyMonkey [110] online survey website. Meanwhile, an open question, which aimed to collect comments about the user interface of IWE or the way of presenting worked examples, was asked after the SUS questions. 23 serious users' responses were collected. The detailed responses are listed in Table 7.2, users with indices 9, 20 and 21 were serious users out of lab, and the rest were serious user in the lab.

User Index	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS
1	5	1	5	1	5	1	4	1	5	1	97.5
2	5	1	5	1	4	2	4	1	5	1	92.5
3	4	1	5	1	4	2	5	1	5	1	92.5
4	4	1	4	1	4	2	5	1	5	1	90.0
5	5	2	5	2	5	2	5	1	5	2	90.0
6	5	1	4	1	5	2	5	3	5	1	90.0
7	4	1	4	1	4	1	5	2	5	1	90.0
8	5	1	5	1	4	2	5	1	4	2	90.0
9	3	1	5	1	4	1	4	1	5	1	90.0
10	5	1	5	1	3	1	5	5	5	1	85.0
11	5	1	5	1	4	2	5	4	4	1	85.0
12	5	1	4	1	4	2	5	3	4	1	85.0
13	3	2	5	5	4	1	5	1	5	1	80.0
14	4	2	5	1	4	3	4	2	4	1	80.0
15	5	2	5	2	5	2	4	3	4	2	80.0
16	5	2	4	1	4	2	4	2	4	2	80.0
17	4	2	4	1	4	1	5	2	4	4	77.5
18	3	2	4	2	4	2	4	3	4	2	70.0
19	3	1	5	1	3	4	4	3	3	2	67.5
20	2	3	3	2	3	3	3	2	3	2	55.0
21	3	4	4	2	3	2	4	3	2	3	55.0
22	4	4	2	2	2	4	3	3	2	3	42.5
23	1	4	2	4	2	3	2	4	2	4	25.0
The average SUS Score is											77.8

Table 7.2 SUS Original Data and SUS Score for Each Evaluated User

From Table 7.2, it is shown that the average SUS Score of IWE is 77.8 and its correlation percentile rank is 82.1%. This result can be interpreted as IWE is considered more usable than 82.1% of all products tested using SUS, which means IWE is in the top 17.9% rank of all the products tested using SUS. (The Lookup Table for SUS Benchmark Data published by Sauro [97] is provided in Appendix 13.)

Sauro [97] also released a commercial calculator, which provides easy ways of scoring and representing SUS results. By using this calculator, raw SUS scores can be converted into percentile ranks. Meanwhile, all the necessary statistical calculations from comparisons to sample size estimation are included. The IWE usability evaluation result shown in Figure 7.1 was generated by using this calculator. The standard deviation of this SUS score is 18.0, which is based on a sample size of 23, with 95% confidence level and its margin of error is 8 points. Even consider the lowest possible SUS score of IWE, which equals to 69.8, it is still above 68.

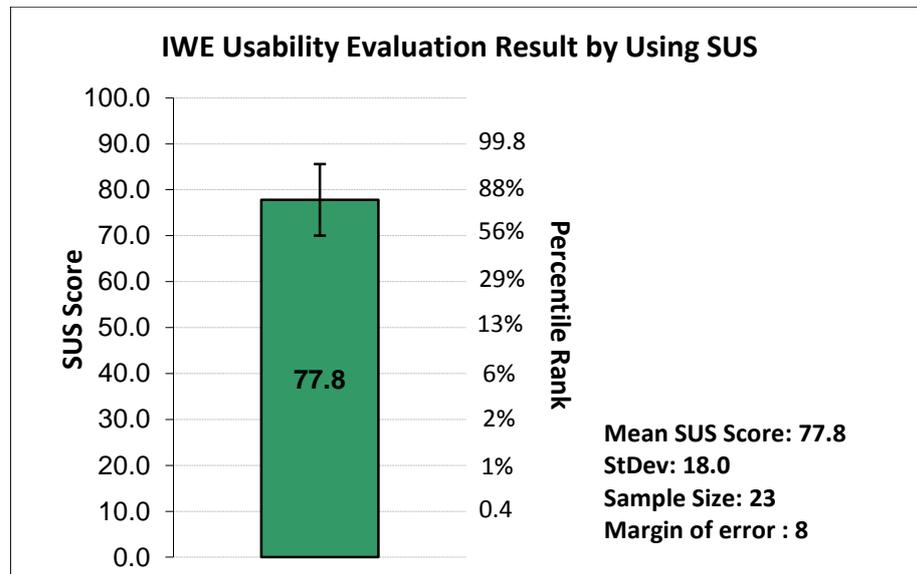


Figure 7.1 Graphical Representation of IWE Usability Evaluation Result

7.4 Utility of IWE

7.4.1 A Conceptual Discussion

The SUS score shows the usability of IWE is acceptable, but this does not measure the utility level. However, the deployment of the IWE is motivated by educational purposes, so IWE must not only allow students to use the tool effectively; it must also allow students to explore worked examples in ways that the literature suggests will support learning. This section will discuss whether the tool is effective at the utility level by analysing other collected data. The expectation of how students will use worked examples will have been borne in mind by teachers when they created them. Hence, it is necessary to figure out whether the way students used these worked examples matches the teacher's expectation.

IWE used a logging technique to record the interaction between the user and the software. The data was collected unobtrusively without influencing the user's working style. Hence, it is possible to reconstruct what the student was doing and the time spent on each steps of the worked example. The usage traces generated from log data can demonstrate how students actually use these interactive worked examples. Through comparing the teacher's prediction of usage and the students' actual usage, it is possible to argue the effectiveness of IWE at the utility level. The detailed process of investigating the effectiveness of using worked examples is shown in Figure 7.2 as well as the interpretations.

1. The teacher has a prediction on the use of a worked example. For example, the allocated time of viewing different steps; some students may be stuck at a particular step and so on.
2. Then, the teacher can look at a trace of usage, which was created from the log file. The actual trace of usage may match his original prediction or not. If similarly matched, it can claim that the tool can deliver a worked example which allows this student to use the worked example in a way that follows the teacher prediction, which could support the student learning. Hence, the tool is effective from the teacher's educational point of view.
3. If not similarly matched, further analysis of how students actually use the worked example should be conducted. For example, the student may have used the worked example in a reviewing situation rather than in a learning situation. They may already have looked this example before, as they can use IWE not only in the lab, but also on their own machine. So the next question is whether the teacher can infer a valuable mode of use based on analysis the additional data, including answers to questions or feedback provided by the students. If the teacher can, the tool is still effective from the teacher's educational point of view; otherwise, the tool may be less effective.

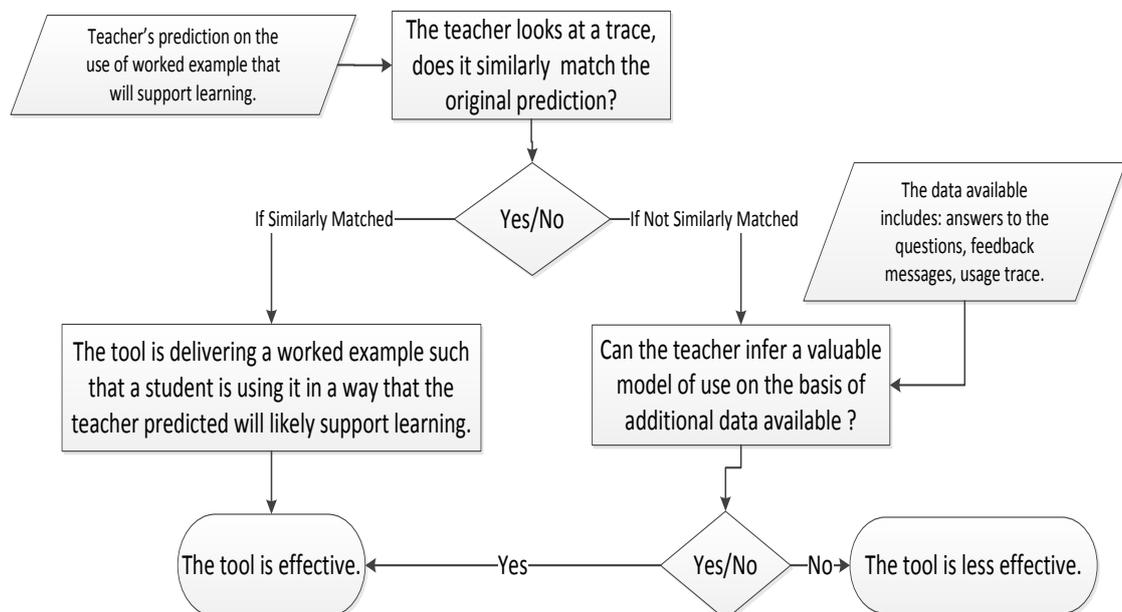


Figure 7.2 The Process of Investigating the Effectiveness of Using Worked Examples

The analysis method described above requires the preparation of data including the teacher's prediction of the use of worked example and the students' usage traces produced from log files. As the judgements of similar match are quite

flexible, it was required to be finished by the researcher and the teacher together. The teacher's prediction will be reported together with the individual worked example usage trace. The usage trace can be produced objectively following criteria, which are described below:

1. It should record the name of the worked example, the time of starting and finishing exploring, the executed step numbers, if the step is a question step, the operations of viewing (by pressing Next Step or Back Step buttons), the time spent on each step and the time spent on interaction activities. The trace does not include the answers to questions; the answers come from looking at the answers files.
2. The time of viewing each individual step can contain useful time (a reasonable length of time spending on a step) and the idle time (a longer session between steps). The useful time for each step should be correlated with the complexity of the step. The idle time can be vary and needs to be identified subjectively, as answering a question step may cost longer time than only looking at the contents in a step. The idle time can also be affect by other reasons. For example, students using IWE in the lab, may have a discussion with the tutor, or try to do an exercise after looking at several steps of the worked example.
3. Any other operations while using the tool, for example, sending feedback messages.

In the following section, several usage traces will be reported. They are analysed accordingly as examples in terms of showing the utility of IWE. The criteria for selecting these usage traces are explained below.

1. The usage traces must belong to those 23 users who filled in the SUS questionnaire.
2. At least one usage trace of *DrawingClockProgram* should be considered. Because it is the first example students used, some interesting results may be found.
3. The worked examples, excluding any *DrawingClockProgram* examples, must contain embedded questions.
4. At least one example from the planned use in a Lab Exercise should be included.

5. At least one usage trace of the special example - *FindErrorExample* should be included.
6. The user must have answered all the embedded questions. If the user sent feedback messages, the content of that feedback is another important aspect to be considered.
7. Other particularly interesting usage patterns, for example, going back and forward several times for a number of steps.

By applying these two different set of criteria, for selecting subjects and relevant traces, several usage traces are constructed and presented. The teacher's prediction of using different worked examples is dependent on the contents and purposes of the worked examples. The purpose of the selected worked example will be described firstly, and the student usage data and teacher's predictive data will be presented together, in order to assess whether the students' usage matches the teacher's prediction, or can meet the purpose of using this worked example.

The purpose of *FindErrorExample* and how to represent the contents of the worked example originally described by the teacher were cited below:

“FindErrorExample aims to help students tackle a particular style of exam question, which involves finding errors in code and giving the appropriate remedy. The typical question gives the problem to be solved, and may hint at the intended algorithm, before giving the number of errors the student should look for, as well as the incorrect code itself. In the worked example, six strategies are proposed for finding errors, and each in turn is applied to an example question. As each error is found, the student is prompted to consider how to write down the error and how to fix it. The example uses three panes, one for the question, one for the strategies, and one for the model answers to the question.”

The usage trace and the teacher's prediction of usage are shown together in Table 7.4. In this table, the first column lists the step of the worked example; the second column lists the operations used to control viewing. The character in the table's second column represents the operations: F means Next Step button is pressed; B means Previous Step button is pressed; S means Restart button is

pressed to review the example again from the beginning; and Q means this step is a question step. The third column lists the time spent on a particular step in seconds. The fourth column lists how many words that subjects need to read in a step. The fifth column lists the number of panels used to show the content of a step. The sixth column lists the teacher's estimation of the time that subjects might spend on a step. The seventh column lists the complexity level of a step, based on the teacher's prediction. The eighth column lists the teacher's explanation of each step.

The abbreviations used in the table are: S = Step Number; O = Operation; T(s) = Actual Time Spending on a Step; W = Words to read (in all relevant panes); P = Panes to look at; ET(s) = Estimating Time (Seconds); C = Complexity is coded as: H (High), (M) Medium, (L) Low.

The *FindErrorExample* usage trace of User index 1 in Table 7.2 and the teacher's prediction of using this example were shown together in Table 7.3. The questions, model answers and user answers to these questions are listed in Table 7.4.

FindErrorsExample							Used at 17 Dec 2013 15:12:15 GMT
S	O	T(s)	W	P	ET(s)	C	Teacher's Explanation of Each Step
0	F	11		1	30	L	Introduction – Explanation of how to use the tool
1	F	1					
0	B	1					
1	F	3					
0	B	1					
1	F	4	85	1	20	L	Aim of the worked example, and what to expect – this style of question
2	F	81	110	2	80	H	The example question – a short description and a short program. Invitation to try solving it themselves
3	F	5	45	1	13	L	Open-ended question on how they got on. Invitation to read on.
4	F	23	200	2	90	H	The 6 strategies are presented. Invitation to try applying them.
5	F	12	120	2	21	L	Invitation to try Strategy 1 – try to work out how the code should be working – usually hinted at in question.
6	F	10	30	2	18	L	Strategy 2: initialization of variables. Invitation to check each variable for initialization
7	F	7	35	3	12	M	stop_word_list – ok – been created and initialized.
8	F	5	40	3	12	M	old_words – ok – it's a parameter.
9	F	8	40	3	10	L	i – ok – this is a for loop variable.
10	F	13	100	1	40	H	words – explanation of how this is an error – invitation to write down the explanation as you would in an exam.
11	F	2	105	4	26	M	Model answer given for words initialization.
12	F	1	50	1	14	M	Strategy 3 – are loops working correctly? Only a for loop here – and explained how this is ok.
13	F	2	105	3	30	M	Strategy 4 – check tests in loops and if statements – invitation to check the if statement
14	F	6	40	1	17	M	More on the if statement
15	F	1	35	1	10	L	More on the if statement – showing in fact that there is an error in the if statement
16	Q	79	25	2	48	H	Question – asking to write the explanation of why the if statement was an error.
17	F	14	90	4	30	L	Model answer and explanation.
18	F	105	70	2	18	M	Strategy 5 – indentation – invitation to check.
19	Q	64	10	2	10	L	Question – yes/no on whether there is an indentation problem – there is.
20	F	10	55	3	11	L	Identification of where the problem is and invitation to identify and write an explanation
21	Q	71	15	2	40	H	Question – write down an explanation of the problem
22	F	16	75	2	25	L	Model answer
23	F	17	45	2	15	L	Strategy 6 – are all computations being performed correctly? Invitation to look at what has not been looked at closely
24	F	23	70	3	28	L	Only line not explored – words = words + i
25	Q	7	10	2	8	L	Question – is this line ok? (it's not, needs an [] around the 'i')
26	F	14	50	3	11	L	Answer – no – invitation to see what is wrong, and try writing it
27	Q	34	10	2	40	H	Question – explain the problem
28	F	50	80	2	23	M	Model answer
29	F	50	60	1	15	L	Closing summary.
Total Student Time:		751	Total Predict Time:		765		

Table 7.3 User Index 1 Usage Trace of *FindErrorExample* with Teacher's Prediction

The teacher also made the following comments while reviewing the example, as self-reflection. In step 2, he believed that many students may not try working out the errors. In step 4, students will not do as suggested, because the task is too big. In step 22, he identified a little mistake in the content of third panel. In step 24, he realized that these presented contents are quite tricky, as he implicitly asked students to make a judgement on how good their answer was to decide whether they got it right or wrong. In step 29, he believed the purpose described here could be shown in the starting step of this example.

Step No.	Question Contents	Model Answer	Student Answer
Step 16	As an exercise, type in how you would explain this error and how to solve it.	The test in the if statement in Line 4 is wrong. We should be testing for whether the current word we are looking at is NOT in the stopword list ? Only then do we add it to our new list. Change the line to: if i not in stop_word_list:	It should be if i not in stop_word_list.
Step 19	Can you see any indentation problems?	Yes	Yes
Step 21	Write down how you would explain the indentation problem with the return statement, and how you would solve it.	The return statement is not indented correctly. Just now the function will return the first time a valid word is found. It should be moved out to the same indentation as the for loop header - that is, it should only be executed once the for loop is finished.	It should be at one indentation level.
Step 25	Is this line ok?	Yes	Yes
Step 27	Write down what you think is wrong with this line, and how to correct it.	The update to the words variable on line 5 is incorrect. A word is being added to a list, but only lists can be added to lists. The line should read: words = words + [i] instead of words = words + i	Should be [i].

Table 7.4 User Index 1 Answers to Questions and Model Answers in *Find Error Example*

Based on the information in Tables 7.3 and 7.4, an analysis of trying to understand this student's behaviour from the usage trace was carried out.

This student approaches the worked example as a test question which he can try to answer, rather than as an opportunity to learn a strategy to solve this kind of question. His goal is to find the right answer to the question, not to learn about how to answer these questions. 81 seconds were spent on reading the problem in step 2, which is long enough to finish reading the problem, or even have extra time for thinking about the answer to the problem. In Step 3, 5 seconds is not enough to think about the open ended question. He only spent 23 seconds on step 4, which is not enough to read all 6 strategies. He seemed to understand

the initialization, looping introduction to the conditional tests, as he skipped through the steps from step 6 to 15 and again ignored the invitation question in step 10. He virtually just pressed the Next button with no reading from step 11 to 15.

Taking his answers to the questions in steps 16, 21 and 27 into account, they are only partially right as he does not include any explanations. When he came to use this worked example, his aim might be to correct the incorrect code only. However, the teacher wanted students to deliver both explanations of the error and the correction of the error. The teacher also expected students to understand how to answer this type of examination question through using this worked example.

The student may engage with the example when required to answer questions, which was inferred based on the time spent on steps. In step 16, he spent 79 seconds. Writing an answer for the “if statement error” can take a while, including the active thinking time. In step 17, a reasonable time was spent on checking the model answer. In step 18 a longer time was spent, as the teacher’s explanation emphasized the need to pay attention to the indentation in the code. So it is possible that this student began to look at the indentation error in the code in this step. Step 19 was a Y/N multi-choice question, but he spent quite a lot of time thinking about this and the previous step 18 was linked to it. Based on his performance from step 16 to 22, it is possible that he was less sure about the test in the if statement (section from 16 to 17) and particularly the indentation and correct code for appending an item to a list (section from 18 to 22).

Step 24 followed by step 25 is another Y/N multi-choice question. In contrast with previous one, the student may answer this question more confidently. After giving the answer in step 27, he spent a reasonable time on the model answer in Step 28. In step 29, reasonable time was spent reading the closing summary which could include looking at the calculation problem.

The previous literature review of faded worked example explained that faded worked-out examples can make the “dialogue” between students and learning objects happen, as it can make the students do self-explanation in order to

engage with learning objects and could help students to arrive the same level of concept engagement as the teacher's expectation more quickly. Although this student did not use *FindErrorExample* as the teacher expected, the "dialogue" still happened after answering the questions. A certain level of engagement happened when he may not be totally confident of the answer. Hence, following the procedure shown in Figure 7.2, the tool was usable and the example was useful to him. Another valuable finding is that the teacher reported that in the written exam, providing only the correction code occurred very often. The way of answering this type of question, either when using IWE or in the written exam, highlighted the need to emphasize the importance of explaining errors in the future. When designing the interactive worked example, it may be necessary to split the first question into two sub questions - explain and then correct; and then reemphasize the need to give an explanation of the error in the second question.

By applying a similar analysis technique, another example of a student's usage trace, who is index 7 in Table 7.2, is given in Figure 7.5. It is about using the *HorizontalLinesProgram*. The presentation structure and overall purpose of this example was cited as:

"The structure is a single main pane with the explanation pane below it. The main pane is used to present both a programming problem and the Python code that solves the problem. The purpose of the interactive worked example is to demonstrate the process of solving this problem - the thinking that goes on during the process - questions to be asking oneself. Whilst at least some of the questions are useful generic questions, the idea that these questions could be reused elsewhere is not drawn out.

The problem was clearly presented in the student worksheet, they were invited to try solving the problem themselves first, and to only use the IWE tool if they were unsure how to proceed. Indeed, they were invited to only go as far through the interactive worked example as they needed until they could see how to continue progressing on their own."

Three out of four questions in this example were answered correctly. His usage trace and the teacher's predictive usage are shown in Table 7.5 (Abbreviations

used here is the same meaning as used in Table 7.3). Answers given by the student are also listed together with the question contents and model answers in Table 7.6.

HorizontalLinesProgram 11 Nov 2013 16:46:13 GMT							Teacher's Explanation of Each Step
S	O	T(s)	W	P	ET(s)	C	
0	F	7	120	1	30	L	Introduction - Explanation of how to use the tool.
1	F	10	45	2	13	L/M	Introduction to the problem. Invited to open the Python IDE so code can be developed alongside. Time can be very short as problem was introduced in the worksheet.
2	F	2					
1	B	1					
2	F	3	30	1	25	M	Explained that it's a drawing program, and invited to come up with what is needed at the start and end of the program. Time can include 10 seconds to think of an answer.
3	F	2	15	2	6	L	Import and complete structure of the code are shown in code window, with a short explanation.
4	F	7	40	2	6	L	Instruction on adding a header comment - shown in code window. - but most are not need to read as comment of the code.
5	F	2	14	1	12	H	Post an informal question - is the solution mainly a loop, conditional or sequence? Time here includes thinking time. For average to weaker students this is likely to be hard.
4	B	11					
5	F	2					
6	Q	21	14	1	10	M	This MCQ repeated pervious infromal question.If student did think about the previous, a quick selection can be made. Otherwise, time should include thinking time.
7	F	4	60	1	12	M/H	Reading the solution. There's little likelihood of students stepping backwards here.
8	F	8	20	1	13	H	Invited to consider what the loop will look like, what is needed, and to hold that in mind.
9	F	3	45	2	32	H	The explanation indicates we're going to explore variable and a loop by combining "create_line" statement.
10	F	19	30	2	17	L	Asked to consider "what will change" in the create_line statement on each new line that is drawn.
11	Q	7	30	1	12	H	The options were all based around the arguments. If step 10 was not well understood, they may continue to this question, and then spend longer here as they repair their understanding.
12	F	10	50	2	21	H	Answer given - the y values change. And then a question is posed as to whether the two y values are the same or different.
13	Q	26	18	1	9	L	MCQ - require to read the options and make selection. If had the answer in mind, selction can be quick.
14	F	2	62	2	24	H	Answer - they're the same. Explain the y value of the start and end for any particular line is the same, although each line will use a different y value from other lines.
15	F	7	50	2	43	H	The y variable in the create_line statement is to be a stepper variable and a while loop are put in place, along with increment to y. Time cost to work out answer could be shifted to the next step.
16	Q	15	13	1	12	L	MCQ - require to read the options and make selection.
17	F	41	100	2	35	M	Answer is given - three values fill into three slots in the code and explanation provided. The three lines in the main pain are highlighted with the correct values inserted.
18	F	15	14	2	39	M	Invited to check whether anything else is required. Time maybe vary.
19	F	38	8	1	4	L	Confirmation that the code is done.
Total Studen tTime:	263		Total Predict Time:		375		

Table 7.5 User index 7 Usage Trace of *HorizontalLinesProgram* with Teacher's Prediction

Step No.	Question Contents	Model Answer	Student Answer
Step 6	Is the solution going to be principally a repetition, a conditional or a sequence? A. Repetition (e.g. while loop) B. Conditional (e.g. if then) C. Sequence (e.g. just a series of steps one after another) D. I don't know	A. The question was whether the solution here is mainly a repetition, a conditional, or a sequence. The right answer for this one is a repetition. Why? Because we're creating 10 very similar lines - the fixed part of the action is the line? It's the same size/dimensions every time - only the position is changing each time.	A
Step 11	We are planning to draw a number of horizontal lines, using <code>create_line</code> . Which arguments to <code>create_line</code> will change on each call? A. <code>x1</code> and <code>x2</code> B. <code>y1</code> and <code>y2</code> C. <code>x1</code> , <code>x2</code> , <code>y1</code> and <code>y2</code> D. Some other combination	B. it's the two <code>y</code> values that changes each time. We've fixed the <code>x</code> values? the start and end <code>x</code> values of 10 and 110, for a horizontal line 100 long - as shown here.	B
Step 13	On each individual call to <code>create_line</code> , are the two <code>y</code> values the same, or different? A. The same B. Different C. I don't understand the question	A. They are the same. We're drawing horizontal lines, so the <code>y</code> value of the start and the end is the same. So we can fix this in our <code>create_line</code> call, as shown here - where we've used just one variable name for both <code>y</code> values.	A
Step 16	What 3 values should replace the three ?? segments shown above? A. 1, 10, 1 B. 10, 100, 10 C. 10, 110, 10 D. I don't know	B. We start at <code>y = 10</code> (starting at (10, 10) remember). The lines are 10 apart, so the update is to add 10 each time round the loop. We'll therefore have drawn ten lines when <code>y</code> is at 100. Once <code>y</code> is greater than 100, we should stop, hence continuing while <code>y <= 100</code> .	C

Table 7.6 User Index 7 Answers to the Questions and Model Answers in Horizontal Lines Example

Based on the information in Tables 7.5 and 7.6, a reasonable conclusion can be made. This student used the example as the teacher expected. Reasons for giving this conclusion are explained below.

The time the student spent on each step matched quite well with the teacher's prediction. He spent very few seconds on reading the problem description in step 2; this could be because this was clearly presented in the lab worksheet. He did engage with the example, as the time spent on step 5 was about 10 seconds short, which meant he did not think about the informal question. But an extra 11 seconds were spent in step 6 for thinking when the same question which was proposed formally and he did answer it correctly. He answered the first 3 questions correctly, so less time was spent on the model answer steps. The fourth question was answered incorrectly, so a slightly longer time was spent on the model answer step compared with teacher's expectation. This student used

IWE in a way that the teacher predicted will be likely to support learning. So IWE was effective for this student, following the investigation process in Figure 7.2.

Other simple analyses of usage only based on students' usage traces and their feedback are described below. Combining user logging information and feedback messages can also provide meaningful information, which can demonstrate the IWE is usable from an objective and subjective point of view.

The usage trace of user index 5 in Table 7.2, viewing *DrawingClockProgram*, is shown in Table 7.7. The usage trace shows that he succeeded in viewing the worked example from the beginning to the end, but might have been confused in steps 14 and 15. This student believed IWE was useful, as he commented that "The worked examples in IWE are presented perfectly clear! It really helped me a lot to understand some of the task I didn't get in the lectures."

		DrawingClockProgram																		11 Nov 2013 16:39:21 GMT								
Step Number		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	14	15	16	17	18	17	16	15	16	17	18
Operation		F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	B	F	F	F	F	F	B	B	B	F	F	F
Time(seconds)		2	1	7	92	3	4	56	17	35	7	6	43	27	17	5	58	125	3	8	8	4	1	2	24	1	2	303

Table 7.7 Index 5 User's Usage Trace of Viewing *DrawingClockProgram*

The usage trace of user index 8 in Table 7.2, viewing *RandomLineProgram*, is shown in Table 7.8. By looking at his answer file for this example, the question in Step 4 was answered correctly, but the answers to the questions in step 11 and 13 were incorrect. His trace shows that a longer time was spent in both the question step (step 11 and 13) and the solution step (step 12 and 14). It is possible that this user actively thought about the question and tried to figure out why his answer was wrong. A long time was spent on step 6, which highlighted a basic piece of code about writing a while loop. It seems that he did need more support in learning how to program. This worked example may be too hard, so it is not useful to him. His comment that "[there were] not enough of them." suggests that he did feel that worked examples were valuable. In this light, the long pauses suggest that he was working hard rather than that the usability of the system was slowing him down.

		RandomLineProgram										11 Nov 2013 16:19:32 GMT						
Step Number		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operation		F	F	F	F	Q	F	F	F	F	F	Q	F	Q	F	F	F	F
Time(seconds)		14	22	23	20	20	19	109	6	38	73	4	173	163	26	241	5	17

Table 7.8 Index 8 User's Usage Trace of Viewing *RandomLineProgram*

From these two usage traces, it was objectively shown that users can navigate through interactive worked examples successfully. A reasonable time was spent on each individual step. More time was spent on question steps and model answer steps. If a question was answered incorrectly, reasonable time was spent on the solution step.

The responses to the open question attached after the SUS questionnaires are subjective feedback messages from users. These users' subjective opinions can also demonstrate IWE is usable and useful. Some positive comments were obtained from users in Table 7.2 who gave IWE higher SUS scores. For instance,

- User index 1 mentioned that “The IWE was an extremely useful tool for learning programming. I highly advise that it is used with future groups and over programming course”.
- User index 6 mentioned that “The user interface as well as the way of presenting the worked examples is fine and quite easy to get around in. The only thing that could be improved just a bit is the overall speed and performance of the tool.”
- User index 13 mentioned that “I believe that worked examples are the best way in which I personally learn to do something. Therefore the IWE system is extremely helpful for studying and helping me to follow the steps and thought processes involved in programming”.

The data presented above demonstrates that students believed they benefited from using IWE and they liked to use these examples delivered by IWE.

Modifications were made following the suggestions in section 6.7.1.1, and no problems were reported with using the revised user interface. Hence, as a software tool, IWE is usable from the HCI point of view and it has utility as an educational software tool.

7.4.2 Possible Reasons for Giving IWE a Low Score in the SUS Questionnaire

There are 5 individual responses in Table 7.2 giving IWE a lower than average SUS score. Based on the data resources of this field study, including answer files, feedback messages collected by using IWE, log files of using IWE, and the open question answer of the online survey, some possible reasons for these low scores can be suggested.

For example, user index 20 replied to the open question after the SUS questionnaire that “I don't think I gained a lot from the IWE. Maybe if it had focused smaller - e.g. here is how a function works - a few examples of how you name, define and return them. I think the tasks were too big for python. It was meant to be a course for those with no programming experience, so it should have focused on achieving smaller tasks.”

User index 21 replied to the open question after the SUS questionnaire that “I think it would benefit maybe to have a workshop or tutorial or something to help (especially beginners) out with the practical programming part of the course. This would help immensely.” These two responses can be interpreted as the level of Intrinsic Cognitive Load of worked examples in IWE is too high for them.

Evidence was found to support this interpretation from users' feedback messages sent using IWE and the usage traces of viewing worked examples which were produced from log files. For instance, the user index 20 sent 4 messages in turn using e-mail, listed below. The first 3 were sent while using IWE in the lab on 26 November 2013 and the fourth one was sent through e-mail on 27 November 2013. This indicated that this student was not only using IWE in lab, but also used it on his own computer out of the lab.

1. Copied code but doesn't print anything. [*ListProcessingFunctionExample*]
2. My mistake, mistyped and wrong indentation.
[*ListProcessingFunctionExample*]
3. I'm not sure about for example def capitaliseNew (words): - where these terms should come from though understand that it is function (parameter). I think string library should be explained/taught. Also think there should be a more explicit connection with Alice and Python so that can build on that, rather than feel as if this is a sudden foreign language to learn. [*ListProcessingFunctionExample*]
4. I didn't do overly well... Think I need to put a lot more time in to truly understand python. [*ListProcessingFunctionExample*]

The fourth feedback message shows that the user index 20 did have trouble in learning programming when using the Python language. This indicates that the previous interpretation is accurate.

The user index 20 usage trace, which is shown in Table 7.9, also matched this student's feedback message. He spent reasonable time viewing *ListProcessingFunctionExample* in the lab and did benefit from using this worked example. In Table 7.9, it shows that the student looked at the example twice, and was confused about the content in step 9, as in the trace table, he looked up and down around step 9 for 3 times. This information can also be found from his third feedback message. He tried to practise with code from the existing worked example to build up his programming experience. He spent a long time viewing steps 18 and 19 the second time he viewed the example and successfully ran the code for the example. This was claimed in his first feedback message and confirmed in his second message.

In this trace, the time spent on steps 12 and 13 shows that he did think about the question and read the answer carefully. Based on his answer files for the two times he explored the example, he answered the question in step 12 wrongly the first time, so he read the answer carefully; the second time of viewing, he quickly passed through these question and answer steps.

		ListProcessingFunctionsExample																		26 Nov 2013 09:22:35 GMT								
Step Number		0	1	2	3	4	5	6	7	8	9	8	9	10	9	10	11	12	13	14	15	16	17	16	17	18	0	1
Operation		F	F	F	F	F	F	F	F	F	F	B	F	F	B	F	F	Q	F	F	F	F	F	B	F	F	S	F
Time(seconds)		19	16	289	9	10	5	6	2	4	1	9	108	59	7	161	17	84	85	231	120	141	1	1	0	15	1	15

		ListProcessingFunctionsExample																		26 Nov 2013 09:46:32 GMT				
Step Number		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	17	18	19	
Operation		F	F	F	F	F	F	F	F	F	F	F	F	Q	F	F	F	F	F	F	F	B	F	F
Time(seconds)		1	1	0	0	0	1	0	0	0	1	0	0	1	2	1	0	0	0	3	3	446	544	

Table 7.9 Index 20 User's Usage Trace of Viewing *ListProcessingFunctionExample*

User index 21 usage trace of the *HorizontalLinesProgram* is shown in Table 7.10. This example contained 4 questions. From this student's answer file, it was shown that he answered 3 questions incorrectly; he only answered the first question correctly. This matched his usage trace, as he spent more time on the solution step (including stepping backwards to review the question), except the first question's answer. Given the wrong answers and the time spent, this trace helps to explain his claim that extra workshop or tutorial should be given for the practical programming part of the course.

		HorizontalLinesProgram																		11 Nov 2013 14:15:21 GMT				
Step Number		0	0	1	2	3	4	5	6	7	8	9	10	11	12	11	12	13	14	15	16	17	18	19
Operation		F	S	F	F	F	F	F	Q	F	F	F	F	Q	F	B	F	Q	F	F	Q	F	F	F
Time(seconds)		23	1	1	1	46	1	1	392	11	1	38	22	21	2	45	141	43	52	78	111	50	2	9

Table 7.10 Index 21 User's Usage Trace of Viewing *HorizontalLinesProgram*

The *HorizontalLinesProgram* usage trace of User index 23 is shown in Table 7.11. This student's answer file shows that he answered all 4 questions correctly. The time spent on the solution steps was very short. It is possible that this example was too simple for him. His expected support from using the tool was different from what IWE offered, so a very low score was given.

HorizontalLinesProgram 11 Nov 2013 14:26:23 GMT																										
Step Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	14	13	14	15	16	15	16	17	18	19
Operation	F	F	F	F	F	F	Q	F	F	F	F	Q	F	Q	F	F	B	B	F	F	Q	B	F	F	F	F
Time(seconds)	2	5	1	5	3	1	9	2	16	5	9	26	2	59	5	7	4	39	4	3	2	4	32	13	17	48

Table 7.11 Index 23 User's Usage Trace of Viewing *HorizontalLinesProgram*

The users' usage traces for those students giving a low SUS rating show that these users did use IWE after the lecture for their individual purposes, either to finish the lab exercise or to prepare the lab exam.

Based on the discussion above, some possible reasons for users giving IWE a low SUS score can be identified. Whether the content of the worked example fits with the student's learning level is the most likely reason that affects the SUS score they gave for IWE. If students believed an example was too simple or they did not learn something after using the tool then they gave IWE a low score, as Intrinsic Cognitive Load of the worked examples is too low for them. From the education point of view, they are already above the level of novices, other techniques for scaffolding their learning should be used for them.

Some light users' responses on why did not use IWE very often also confirmed this conclusion. For example, several students mentioned that:

- "I did not find the tool to be very helpful as often I would only want to consult it for a specific part but would not be able to skip through to the part that was needed. I also found it to be quite limited in content and the content there was well explained in the lectures."
- "The exercises were simple enough to be completed without IWE."
- "I used the IWE teaching tool whenever I had a doubt about how to do something or a better way of doing it. I did not use it more often as I didn't need to."
- "I personally found CS1-CT course very easy. If I would have encountered some problems, I would probably use IWE teaching tools."

- “I never really felt I needed much more help, however if there was something I was having trouble with I'm sure I would have used it to get a better understanding. I felt that we covered everything in detail in the lectures.”
- “The IWE teaching tool is great in my opinion. However, I used it only when I got stuck in writing some of the examples and the problem was that in order to get to the point where I am stuck I have to go through the whole program and answer question about "easy stuff".”

If users believed the example was too hard, they could also give IWE a low score. These examples were above their level, which can cause cognitive overload to these students. For example, the user index 20 and 21 in the serious user group. From the education point of view, these worked examples are useless to them; more support should be offered to help them to understand the concepts firstly.

It can be concluded that, from software engineering point of view, the usability of IWE is good. Based on the log trace analysis of students' activities while using these interactive worked examples, it was shown that students were able to use IWE as expected. Given the match between a teacher's expectation and a student's actual use, these worked examples created by IWE are likely to be useful to the students, which is a good starting point for building valuable education software.

7.5 Teacher's Evaluation Result

7.5.1 Evolution of the *HorizontalLinesProgram*

After finishing the lab session in 11/11/2013, students' answers to the embedded questions in the first batch of released worked examples were provided to the CS1CT lecturer. Based on the example answer summary report provided, the teacher made some modifications to the worked example of *HorizontalLinesProgram* and *RandomLineProgram*. These modified examples were used by students in the lab session on 12/11/13. Here *HorizontalLinesProgram* is selected as an instance to describe the process of evolving the worked example. The teacher found that the number of correct answers to questions 11, 13 and 16 was lower than expected. So he decided to add more hints into the questions. Tables 7.12 and 7.13 list the answer summary

before and after modifying the question, and each individual question comparison results were shown in Tables 7.14, 7.15 and 7.16. The green colour highlighted the right option that should have been chosen to answer the question. For questions 11 and 16, option D is “Don’t Know”, but for question 13 option C is effectively “Don’t Know”, as there only three options.

These 3 questions aimed to test students understanding of some basic concepts of programming in Python. Question 11 tests whether students understand how to use the `create_line` function to draw a single line. Question 13 tests whether students can identify how to draw numbers of horizontal lines using the `create_line` argument, focussing on whether the “y” values of the start and finish points change. Question 16 tests how the while loop, to draw 10 horizontal lines, is executed based on the terminating condition.

	A	B	C	D	Right %
Question 11	3	13	10	0	50%
Question 13	8	10	8	N/A	31%
Question 16	0	7	13	6	27%

Table 7.12 Answer Summary before Modification

	A	B	C	D	Right %
Question 11	2	4	5	1	33%
Question 13	8	4	0	N/A	67%
Question 16	2	2	7	1	17%

Table 7.13 Answer Summary after Modification

Answer	Original Question:	Modified Question:
	We are planning to draw a number of horizontal lines, using <code>create_line</code> . Which arguments to <code>create_line</code> will change on each call? A. x1 and x2 B. y1 and y2 C. x1, x2, y1 and y2 D. Some other combination	We are planning to draw a number of horizontal lines, using <code>create_line</code> . Which arguments to <code>create_line</code> will change on each call? Think carefully here - if you were drawing these lines one after another, do the x values of the start and end point change? Or is it the y values that change? Or both? A. x1 and x2 B. y1 and y2 C. x1, x2, y1 and y2 D. Some other combination
A	3	2
B	13	4
C	10	5
D	0	1

Table 7.14 Answers to Question 11 comparison before and after modification

Answer	Original Question:	Modified Question:
	On each individual call to <code>create_line</code> , are the two y values the same, or different? A. The same B. Different C. I don't understand the question	On each individual call to <code>create_line</code> , are the two y values, y1 and y2 the same, or different? Think carefully about the nature of horizontal lines... A. The same B. Different C. I don't understand the question
A	8	8
B	10	4
C	8	0

Table 7.15 Answers to Question 13 comparison before and after modification

Answer	Modified Question: What 3 values should replace the three ??s shown above? A. 1, 10, 1 B. 10, 100, 10 C. 10, 110, 10 D. I don't know	Modified Question: What 3 values should replace the three ??s shown above? Think carefully about how many times the while loop will repeat with each different set of values... A. 1, 10, 1 B. 10, 100, 10 C. 10, 110, 10 D. I don't know
A	0	2
B	7	2
C	13	7
D	6	1

Table 7.16 Answers to Question 16 comparison before and after modification

There is one point that needs to be noticed. The number of answers in the two groups is different. In the first day's group, the sample size is 26 and in the second day's group the sample size is 12. However, all subjects are in the CS1CT course and are expected to have similar levels of knowledge to answer these questions. The only difference between the two groups is the question content. Meanwhile, the students' answers are a frequency distribution and not a normal distribution. The comparison between the two groups is to test whether rewording questions can improve students' performance in answering questions correctly. So the Chi-square test can be applied and the data can be analysed by using a 2x2 contingency table. However, the small sample size of the second day's group may affect the power of the Chi-square test to detect significant differences between the samples.

Based on the two groups' answers given to the question 11, the Chi-square test results ($\chi^2 = 0.923 < 3.84$ with 1 degrees of freedom and the one-tailed P-value = $0.1684 > 0.05$) shown that the hints added did not work as expected, which means it did not significantly change students' performance in answering questions correctly. It was probably because the hint was too long and caused students' confusion. Students may also consider the whole argument in `create_line` needs to be changed every time, although the value of `x1` and `x2` are the same. So even with the hint provided, the wrong option is still chosen. A possible way of rewording could be "Consider the function call `create_line (x1, y1, x2, y2)` itself, which of the values (`x1`, `y1`, `x2`, `y2`) will be changed on each call?"

Alternatively, the reason the rewording of question 11 did not work might be explained from the angle of cognitive load theory. The intrinsic cognitive load of

this question is too high, which results in this question being too complicated for novices. It is too quick to jump into practice without scaffolding novice students to think about the nature of horizontal lines. In fact, this question combines two questions together, which are how to draw a set of horizontal lines on paper based on understanding the attributes of lines and how to write the implementation code to represent the this way of drawing lines. There is a median step between the two questions which is to map the nature of understanding of the lines attributes into a way of computational thinking. It is a typical programming question, which is modelling the nature of an object and then doing abstraction to map the natural attributes of the object in a way of computational thinking to implement the object in a programming language. For example, for this question, students need to clearly understand how to draw horizontal lines on paper first, including the length of lines, the position of drawing these lines and the spacing between these lines; and then do the computational thinking as well as writing the code in Python.

Based on the answers given to question 13, emphasizing “y1 and y2” was worthwhile. No students selected answer (c), which is “I don't understand the question”. Meanwhile, the number of correct answers to this question was increased significantly, from 31% to 67%. Based on the Chi-square test result ($\chi^2 = 4.34 > 3.84$ in the table of Chi-Square distribution for 1 degree of freedom, the one-tailed P value = $0.0186 < 0.05$), it was shown that subjects who answered questions on the second day have significant difference of giving the right answers compared with subjects performance from the first day. The rewording did improve students' performance in answering questions correctly. In other words, it was shown that students were confused by the question rather than they did not understand the concept behind the question. Another possible reason for the improvement in the students' performance could be the hint asking students to think carefully about the nature of horizontal lines had an effect. This hint promoted students to think about drawing horizontal lines on paper, and helped them to realize that the features are: the value of (x1, y1) defines the starting position of a line and the value of (x2,y2) defines the finishing position of the line. For a single horizontal line, the difference between the value of x1 and x2 defines the length of a line, and the value of y1 and y2 must be equal. To draw a second parallel horizontal line from (X1, Y1) to (X2,

Y2), X1 and X2 must be the same as x1 and x2; Y1 and Y2 must be equal but different from y1 and y2. This difference between Y1 and y1 defines the distance between the two lines. To draw a set of parallel horizontal lines, this process needs to be repeated as often as required. The better result can be understood as scaffolding students helps to map the computational thinking to the nature of horizontal lines. Modifying question 11 into 2 sub questions and introducing the computational thinking early before question 11 may give even better results for answering this question.

Based on the two groups' answers given to question 16, 76.3% (29 out of 38) of subjects chose the wrong answer; of these 29 subjects, 70.0% (20 out of 29) of them chose option (c) as their answer. The lecturer added more hints to the question, however, based on the square test results (Chi squared $\chi^2 = 0.478 < 3.84$ with 1 degrees of freedom, the one-tailed P value equals $0.2447 > 0.05$.), the modification did not work as expected. It was probably because students did not understand the while loop termination condition. In fact, misunderstanding the termination of a loop is a very common error in computing programming education. It is a typical learning problem, as the Jeliot 2000 program [112] was designed to help students to learn and overcome it.

A possible modification plan is described below:

1. One more question can be asked between step 7 and step 8. As in step 7, the explanation of drawing horizontal lines pointed out the only change of these lines is the position. It is a suitable time to ask a question which can promote students to think about the nature of horizontal lines and to scaffold them to do the computational thinking.
2. Add one more sentence in question 11, which aims to reemphasize the needs to think carefully about the nature of horizontal lines.

By quickly using students' answers as a kind of feedback, the *HorizontalLinesProgram* worked example was evolved. The teacher can see how well students understand the concepts introduced in the classroom, based on the worked example usage data. The two days of results clearly showed that how to terminate the while loop was a problem, so the lecturer needs to explain more about how to set up a condition for terminating a while loop in Python.

7.5.2 Lessons Learned From *FindErrorExample*

FindErrorExample was designed to support a particular type of question which would be included in the written exam. It was created during the dynamic ongoing progress of the course. Based on answer files received from students using this worked example, an interesting problem was found. Students often only focussed on correcting the incorrect code, which was not expected by the teacher, as in the written exam, students will lose marks if no explanation of a correction is provided. In the *FindErrorExample*, 21 out of 31 (67.7%) subjects only gave corrections and did not provide the explanation part, so the teacher should emphasize that providing the explanation of correction is very important.

A possible modification plan is described below:

1. At the beginning of this worked example, add some highlighted sentences to explain how to get full marks for answering this type of question.
2. Split the first find error question in step 16 into two sub questions, one is to explain the error and another to show the correct code.
3. Add one more sentence to reemphasize the need to give an explanation of error to the second find error question in step 21. For example, the sentence could be “Remember that you need write down the explanations of the identified error and the correction of the error code to get full marks”.
4. The third find error question in step 27 could be kept as same as before.

7.6 Summary

In the usability review section of chapter 3, it was concluded that a traditional usability evaluation of educational software was necessary, but not enough. The utility of the educational software also needed to be evaluated. Due to the time limitation and the current development stage, a formal experiment in supporting educational effectiveness was not conducted. The educational effectiveness of using IWE is based on discussing and understanding the current worked examples research recommendations and guidance. These worked examples were designed following these recommendations and guidance, and they were also applied following the best practices suggestions for using worked examples. However, in the future, it would be worthwhile to conduct an educational experiment to

measure how much the learning outcomes for students can be improved by using the interactive worked examples delivered by the tool.

A field study aimed to investigate the usability of the tool was conducted within the real educational context. Level-1 students registered in CS1CT course of computing science subject were selected as the test users, as they are ideal novices without previous programming experiences. A combined set of methods of usability evaluation was applied to collect the quantitative and qualitative data. Based on the data collected from students and teachers, a conceptual level investigation of how useful IWE is in a real education context was also addressed. All this evidence demonstrates that IWE is usable for both students and teachers.

The SUS result shows that user interface for students is acceptable, which meets the requirement of designing educational software from the cognitive load theory point of view. The Extraneous Cognitive Load is not an issue for students using the tool to explore or interact with the worked examples. From the open question answers added in the SUS questionnaire, it shows that some students do benefit from using these worked examples delivered by the tool; they liked these worked examples delivered by IWE. The log traces, embedded question answers from students and feedback from students showed that the worked examples are usable and useful to the students, based on a conceptual discussion of whether students' usage of the examples matched the teacher's prediction of usage. Although the user interface for teachers needs to be improved by adding more facilities to create examples more easily, a highly motivated teacher was able to use the IWE to create target worked examples to fit in the dynamic on-going course progress and the delivered worked examples were able to be evolved based on the feedback collected by the tool. Further possible modifications to improve the worked examples were also suggested based on applying Cognitive Load Theory. For novices, they do need more support and scaffolding to develop computational thinking, which also matches the teacher's goal of designing this course.

By using IWE, the CS1CT course teacher could quickly create worked examples to fit in the dynamic on-going course. The teacher was also able to modify the examples, after problems were identified from feedback. The prototype of IWE

is usable to the students and teachers; the education value of IWE is also able to be predicted by the teacher, so it is a good starting point for building valuable education software.

Chapter 8 Discussion and Future Work

In this chapter, final conclusions about the results of this research work are made through discussion of the evaluation results. Section 8.1 discusses how this research verifies the thesis statement. Section 8.2 discusses limitations of the current research prototype system. Section 8.3 suggests changes that could be made to improve this system and possible future work in this research area.

The thesis presented the design, implementation and evaluation of an authoring environment for creating interactive worked examples. In the introductory chapter, the thesis statement was presented as follows.

A usable authoring environment for delivering interactive worked examples can be developed that:

- a. delivers best practice interactive worked examples to students in a computing science context;*
- b. enables teachers to create such interactive worked examples without having to engage in bespoke programming;*
- c. facilitates evolution of them on the basis of feedback from the students.*

This thesis statement requires that the new authoring environment (IWE) must be usable by both teachers and students. So IWE was evaluated within the educational context of a class of problems in computing science. These problems involve transforming one representation of a problem into another related representation. The transformation process is heuristic, not algorithmic, and requires suitable judgements to be made. The worked examples aim to demonstrate how to make suitable judgements for novices, enabling them to develop the ability to make similar judgements on new or unseen problems, for example, transforming a requirements document into an ER diagram or a Use Case diagram.

Based on the results of a wide literature review, no tool was found for authoring these types of interactive worked examples. Tools were found to create lecture videos or animations following an algorithm, or on-line intelligent tutors that try to model both expert and novice behaviours. IWE sits between these however, modelling only the expert's thinking process of problem-solving.

The cognitive challenge for learners to initially develop heuristically-based expert behaviours is recognized, and the need for novices' sustained exposure to the expert's problem solving model is also identified. IWE appears to be the first attempt to fill this need.

8.1 Verifying the Thesis Statement

This research was carried out with the purpose of verifying the thesis statement. To do so, three research questions were identified. Detailed discussion on the answers to these research questions is summarized below.

Question 1. Can an authoring environment be built to support the production of best practice interactive worked examples?

A new authoring environment, named IWE, was developed using the Java programming language. IWE is a single system, which contains two different user interfaces. One is for teachers to create and modify interactive worked examples; the other is for students to explore these examples. XML technology was used to save the resource data of worked examples. The SAX parser was selected to interpret the XML files for the system. The InfoNode [107] docking windows framework was used as a start point of building the main user interface for both teachers and students and the general style of the User Interface was based on the Eclipse user interface.

IWE supports the functionalities associated with best-practice guidance, as identified in Chapter 2:

- Dynamic step-by-step guidance to explain the whole process of problem-solving is supported by the *process* function. The process player allows exploration of worked examples in a step-by-step fashion under students' control.
- Support the Guideline of Self-Explanation Elicitation. The question function supports the creation of a form of faded worked examples, encouraging students to do self-explanation rather than just work through the examples step by step.

- Support the Help Guideline. The explanation function supports the integration of the teacher's explanations into every step, assisting self-explanation and the correction of wrong anticipations. The explanation view is designed to show these useful explanations in order to help students to understand the changes occurring in a step.
- Support the Easy-Mapping Guideline. Highlighting allows the teacher to link related components in different representations, for example mapping part of a problem statement to the relevant part of a solution. The correspondence function allows students to explore relationships between representations by selecting a component in one representation and seeing the related parts of another representation highlighted.
- Support the Structure-Emphasizing Guideline. Documents constructed from fragments allow only revealing required contents. Structure was shown using selective revealing and highlighting of representations, for example, parts of problem statements and high-level and low-level aspects of solutions.
- Support the Meaningful-Building Block Guideline. The highlighting function also allows labelling of a sub-goal during the demonstration. For example, building blocks can be emphasised in ER examples aimed at developing understanding of mappings for different types of relationships and in programming examples identifying the use of patterns.
- Different worked examples can be viewed in different tabs allows students to do comparisons between several examples.
- Faded worked example guidance is also supported. This is addressed by providing two types of questions, which are descriptive questions and multi-choice questions. The answer to the embedded question is always provided in the next step after the question step. So students can get immediate explanations of their judgements. Both an explanation of a just-completed worked out step and an anticipation of the next step can be supported using this mechanism.

Question 2. Can the authoring environment be used to deliver interactive worked examples?

This question can be divided into two sub questions, which are: can a teacher use IWE to create interactive worked examples; and are these worked examples usable by the students. Table 8.1 summarises all the raw data / evidence, collected from the different evaluations and the field study, which will be used to address these questions.

	Effectiveness	Efficiency	Satisfaction
Teachers	<ul style="list-style-type: none"> • The worked examples were created without programming. • Teachers have created worked examples as specified in the literature. • Lab tutors' feedback messages from the one-to-one usability investigation. • Similar worked examples which have already been well evaluated can be reproduced by using IWE, e.g. an ADbC example has been replicated. 	<ul style="list-style-type: none"> • Expert Evaluation (pilot test teacher) for improving the document structure of building worked example. • Interactive Worked Examples were both evolved and created as a course progressed, during the field study. 	<ul style="list-style-type: none"> • A report written by a teacher on using the tool.
Students	<ul style="list-style-type: none"> • This tool delivered educationally valuable worked examples, as specified in the literature. Students used these examples under the best practice recommendations. • Students were observed one-to-one using the tool. • Log file traces showed that students used the examples as the teacher expected. • Based on several previous usability tests, HCI expert evaluation and iterative development, modifications were made to improve the effectiveness. For example, remove play button, and restructure the panels for student UI. 	<ul style="list-style-type: none"> • Learnability in SUS was very high. (Q4 and Q10 results) 	<ul style="list-style-type: none"> • SUS Score. • Feedback messages were received from open questions in a survey or from users of the tool itself. • Interview results from the follow-up one-to-one usability investigation. • Students' feedback messages in the field study.

Table 8.1. Data Resources for IWE Usability Evaluation Conducted during the Development Lifecycle

Question 2a. For teachers, is the system sufficiently usable that they can create examples following the best practice recommendations based on education research results?

The interactive worked example created in the pilot teacher test explained the whole transformation process from a scenario description to Use Cases. The successful experience of building a worked example in a different context showed that the prototype of IWE can be used not only by the researcher, but also by one of the target users. However, using the prototype of IWE it was difficult to build the target interactive worked example, because of the non-intuitive teacher user interface. Suggestions were given to optimize the process of building a worked example, which are reported in section 6.2.4 of chapter 6.

Three teachers from the School of Computing Science in the University of Glasgow used IWE to create 17 transformation-based worked examples in the contexts of a database course and a programming course. All the examples were created within a busy academic teaching semester. All the examples contain the teacher's embedded explanations to each individual step and nine of them are faded worked examples which contain questions between steps. The number of questions embedded is between 2 and 5. These examples are delivered with different purposes, like building students' experience of transformation-based problems, assisting lab practices and supporting exams.

The teachers' evaluation conclusion in section 6.3.4 of chapter 6 provides a positive answer to this question. The teacher's evaluation results in Field study, which was discussed in section 7.5 of chapter 7, also demonstrated that IWE can deliver usable interactive worked examples within a realistic education context. However, their experience of building these examples is worth further discussion.

Application of Worked Examples Design Guidance

The teacher can design effective worked examples, following the guidelines for designing worked example and faded worked example presented in section 2.5.3 of Chapter 2. The tool supports the development of worked examples following these guidelines. However, guidance is not condensed and provided to the teacher; this could be considered for further work. Examples of applying these guidelines are summarised below.

- Faded worked examples with embedded questions were designed to encourage students to self-explain rather than just work through the example step by step. Four faded worked examples used descriptive questions and four used multi-choice questions. *FindErrorExample* combines both types of questions. These examples apply the guidance of Guideline of Self-Explanation Elicitation.
- The teacher's explanations are integrated into every step of a worked example. All the examples delivered were constructed in this way, which follows the Help Guideline. Teachers were able to use several lines of text to explain the reason for making decisions in order to help students to understand the changes occurring in a step.
- All the worked examples produced contained carefully constructed explanations integrated with the examples linking the different representations. Examples also made extensive use of highlighting to identify the relationships between representations within steps and through the correspondence operation. This is an application of the Easy-Mapping Guideline.
- Structure-Emphasizing Guideline is particularly applied in the *Unit-7-Task 3-1* and the *ListProcessingFunctionExample*. The first one used one panel to list five different patterns for writing a loop and used another panel to demonstrate the solution. When a particular pattern is used, the related pattern is highlighted in the other panel. The second one applied this guidance by highlighting a part of the plan and a block of code which implements this part of the plan plus the explanation of how to do this.
- Building blocks are emphasised in the *1-to-1 Relation Demo*, *1-to-Many Relation Demo*, *Many-to-Many Relation Demo* and *FromERToTables* examples, which aimed at developing understanding of mappings for different types of relationships. It also applied in the *Unit-7-Task 3-1* example in terms of identifying the use of patterns. *FindErrorExample* also listed six strategies often used to identify errors while programming in one panel. Through analysing each line of code, question was asked which can be answered based on applying these strategies. Meaningful-Building Block Guideline is typically applied in these examples.

Some interactive worked examples use embedded questions between steps to fade the element of right answer, and encourage students to actively think about the answer to the question. In the next step, the teacher's model answer was provided, which allowed students to compare these two answers, so engagement with the example could happen through the comparisons.

Based on the teachers' evaluation results and the worked examples delivered, the IWE is usable, in that interactive worked examples and faded worked examples, of good enough quality to be presented to students, were created. The teachers reported that the hardest job was to design the worked example rather than implementing them by using IWE, as the conceptual design requires much more intellectual effort. However, there are a lot of existing worked examples available, which can be restructured using IWE. For example, past exam papers are a ready source of examples, as each problem is designed properly, the solution is also carefully prepared and the marking guidelines can help to set up explanations. The following discussion is to introduce the experience of building interactive worked examples based on existing resources.

Experience of Building Interactive Worked Examples

ClassTestRun and *CinemaExPlay* examples were quickly created from previous exam questions, which had model answers and marking guidelines. The worked examples can be divided into two parts, which are from the problem description to the intermediate plan and from the intermediate plan to the code implementation. Three panels were used and questions were set up for guiding the development of intermediate plan in the first part. Mapping the related parts of the plan and code implementation was well explained in the second part.

RequirementToER was built through the collaboration of a teacher and the researcher. This example was turned into a faded worked example by the researcher and another teacher based on a database teacher's detailed PowerPoint presentation and explanations of each step were edited by the database teacher. This example is like reviewing the lecture, as the key points were emphasised using questions and detailed explanations were provided in written format, which are often explained orally by the teacher in the lecture.

This successful experience shows that a teacher's existing hand-out could be another resource for transferring static worked examples into IWE.

DB Relations scenario contains 4 different worked examples, which are *1-to-1 Relation Demo*, *1-to-Many Relation Demo*, *Many-to-Many Relation Demo* and *FromERToTables*. These four worked examples share the same set of documents. The teacher was only required to create two documents and then create four worked examples for different purposes. In this example, the two documents are an ER diagram document, which describes different relationships between entities, and a table document, which contains all tables and columns to represent the ER diagram. The original source of these examples was a database system textbook, which shows that textbooks could also be good resource for building interactive worked examples.

Unit 7-Task 3-1 and *Unit 8 Task-2* were created based on coursework questions and their model answers. *Unit 7-Task 3-1* was a worked example that not only demonstrated the process of problem solving from problem statement to an intermediate plan and then the final Python code, it also identified how the intermediate plan was generated based on programming patterns which were introduced in the lecture. The intention was to help students to observe how the knowledge introduced in the lecture could be applied to solve the target problem. To build this worked example, a teacher prepared two documents: one was used to present the patterns knowledge introduced in the lecture; and the other one was used to describe the problem description, the intermediate plan and the final Python code. When creating the viewing process for students, the teacher indicated the relationship between the intermediate plan and the relevant content in the pattern document. The difficulty of setting up relationship between different representations is confirmed in the teacher's evaluation report when comparing the use of IWE with PowerPoint. These successful experiences show that coursework questions and their model answers can be converted into interactive worked examples.

Unit 8 Task-2 and *PentagonToShapeProgram* were designed to help students to reuse their previous experience to solve a new problem. *Unit 8 Task-2* reused the target document of a previous worked example as a starting point and

showed how a new problem could be solved through reusing and modifying a previous solution. The purpose of *PentagonToShapeProgram* is to show how specialist code to draw pentagons could be made more general, enabling it to draw any regular shape. These successful experiences show that a single document can be used in different interactive worked examples.

Application of Best Practice Recommendations for Using Worked Examples

Trafton and Reiser [40] recommended that worked examples should be offered in a sequence, presenting a worked example which contains problem-solving guidance first, and then providing a similar problem to solve immediately afterwards, then another worked example, then a similar problem, and so on. Presenting a worked example which contains problem-solving guidance first, and then providing a similar problem to solve immediately afterwards is an issue of current instructional design which is discussed in section 2.4.1.5 of chapter 2. By using IWE, this issue can be overcome, as follows.

In the field study, as the teacher recognised the mixed ability of the class, the implementation of using worked examples was slightly modified. Students were asked to solve a new problem first; if they could not then they were invited to view a worked example in IWE for that new problem, if they could then after working out the solution to the problem, they were invited to review the worked example in IWE and to compare their solutions with the example solution. And then they were invited to solve another problem, related to the first. For example, *HorizontalLinesProgram* is provided in IWE, the related problem to be solved is to draw a grid of cells (like an empty table) with 5 cells across and 4 cells down. This is explained and addressed in section 7.1.3 of chapter 7.

Hence, it can be concluded that IWE allows teachers to present examples following the best practice recommendations.

Question 2b. For students, are these delivered examples usable?

Worked examples produced by IWE are useable by students. Based on the log file data from the evaluations and field study, the worked examples delivered were used by more than half of all the students involved. The online survey

questionnaire results and the answer files received through emails all indicated that students did use them on their own machines. After implementing the short-term Improvements to the Student User Interface, which were identified in section 6.7.1.1 of chapter 6, no subject reported issues with using the tool to explore worked examples in the field study.

The log file data also shows that subjects interact with these worked examples step by step in a reasonable time. The online survey questionnaire results showed that the majority of subjects believed using these examples could enhance their understanding of what they learned in the lecture.

The results of one-to-one observation while subjects were using IWE also showed that all the fragments associated with a step were carefully viewed, especially the highlighting parts and the explanations, and most subjects did think about the questions seriously and carefully. Even subjects who did not provide answers to the embedded questions were forced to actively think about the questions and then compare their answers with the teacher's model answers.

In the field study, the result of SUS score presented in section 7.3.2 also shows the tool is usable by students. The collected students' feedback messages in the field study also shows they were satisfied with using the tool.

As IWE is designed for educational purposes, section 3.3 of chapter 3 discusses the need to carry out extra evaluation work to determine the utility of the tool. Several students' usage traces were constructed based on analysing log files in order to determine how they were using the tool. The analysis also made use of students' performance data elsewhere on the course, in order to take account of different ways in which the worked examples may be used. The teacher created an imaginary usage trace based on how he intended the worked example to be used to cause learning to take place, based on best practice principles. Through comparing students' actual use with the teacher's predicted use, a determination can be made of whether learning is likely to be taking place. Some of the students for whom usage traces were constructed certainly did follow the expected use. Other students' usage patterns could be explained when, for example, their position in the class was considered.

Hence, it can be concluded that all these delivered worked examples are usable by students.

Question 3. Can teachers create new examples or evolve existing examples over time to match the dynamic nature of teaching, on the basis of students' feedback and usage data?

The field study result clearly indicates interactive worked examples can be evolved over time based on the students' feedback. A little mistake was found in the *DrawingClockProgram* by students. They sent feedback messages to the teacher, and the teacher was able to modify and update the erroneous part of the example on the same day.

Section 7.5.1 of chapter 7 discussed the evolution of *HorizontalLinesProgram*, based on students' answers to the embedded questions. The students' answers provide data for the teacher to understand students learning progress, which can lead to an adjustment of the teaching in a future lecture. This also shows the education value of IWE.

In the field study, four examples are created to fit the dynamic teaching process, which are *ListProcessingFunctionExample*, *PasswordCheckerExample*, *JustAMinuteExample* and *FindErrorExample*. The first one is designed for supporting the lab exam and the rest are for written exam, due to the teacher's realisation that these parts were not well explained in the lecture. These two examples show the power of the authoring environment for creating interactive worked examples at short notice.

Hence, it can be concluded that teachers can create new examples or evolve existing examples over time to fit in the dynamic teaching process on the basis of students' feedback.

Summary of Answers to Research Questions

The three research questions are answered positively based on the evaluation results. It can be concluded that the thesis statement is verified. The new authoring environment enables the teacher to create usable interactive worked examples, which integrate explanations and questions. These delivered worked

examples are usable by students. IWE also supports the evolution of examples over time based on students' feedback provided.

All the examples created show the power of IWE, which is a general authoring environment for the teacher to create transformation-based worked examples and faded worked examples. Although it is only a prototype, it can be used in a real education context following the best practice recommendations for designing and using worked examples. Potentially, it can be used realistically in other courses of computing science, for example, software engineering is identified.

The field study shows the power of using feedback and log data to evolve the worked examples and teaching practice, data impossible to gather for worked examples delivered using more conventional means such as text books or PowerPoint presentations. IWE can provide data for the teacher to understand students' experience of using the worked examples. These data are particularly valuable for the teacher to make adjustments to the instructional design to meet students' needs. Given the time constraints on a typical teacher, the data needs to be automatically analysed and presented to the teacher for this kind of in-course adjustment to be possible.

There is an extra benefit for using IWE. It is a general authoring environment for both producing interactive worked examples and providing a common presentation environment (one interface for students to learn about) for exploring interactive worked examples. This latter aspect is particularly useful when students are using worked examples in different classes, as they do not need to familiarize themselves with different learning environments.

8.2 Limitations

Many lessons were learned during several evaluations from students and teachers. It is shown that the prototype IWE is a useful tool, which can benefit both teachers and students. However, some problems were also identified. Some problems concern issues of implementation and other problems concern the use of IWE from the educational point of view. With the suggested modification

discussed in section 6.7.1.3 of Chapter 6 and further development to improve the Teacher User Interface, the teacher's experience of using IWE should be able to be improved. Interactive worked examples can be created more easily by the teacher. The following limitations are noted.

- The evidence for the educational value of *interactive* worked examples in computing science is weak. Therefore, it is worth carrying out educational experiments to find evidence which can directly support the educational effectiveness. The research question could be whether students' use of the *interactive* worked examples delivered by IWE indeed leads to a better learning outcome.
- In the field study, the teacher is playing two roles: firstly as 'the teacher for the course' (a role that instructors would usually have in using the system), and secondly as the PhD supervisor critiquing the analysis of the evaluation data (a role that no other teachers would ever have). A mechanism for automatically analysing students' usage of worked examples is worth developing in order to reduce the teacher's workload, given that the usage data can be so valuable for improving the on-going teaching and learning process.

8.3 Suggested Structure of Future Work

Based on the discussion above, the power of the prototype of IWE for both teachers and students has been demonstrated. Although some problems were identified during the evaluations, the possible actions to overcome these problems were very clear to the researcher. IWE only opens a door for teacher to author powerful interactive worked examples in a low cost way; much further work remains to be done to improve it. This section will analyse the possible future work which needs to be done in order to enable further research that can contribute to computing education and education using computing technology.

8.3.1 Analysis of Possible Future Work

Improving the Teacher User Interface

An improved teacher user interface could be implemented but this would require significant effort, as the suggested changes described in 6.7.1.3, are more complex than those made to the student user interface before the field study. Many of the suggested changes for the teacher interface are moving towards commercialisation of the tool. The major changes include: increased functionality for creating explanations, defining documents, defining processes and flexibility of document modification after a process has been defined.

- Providing more functionality in the explanation panel would allow teachers to more clearly link their explanations to the document panes by using different text styles to add emphasis and different fonts to differentiate problem text from program constructs, for example. It would also be valuable to be able to define links to other documents, such as tutorials, or to Web resources.
- The actions for enhancing graphical document construction through using dragging and dropping and improving the text formatting while creating textual documents.
- The actions for improving the definition of processes, including creating composite fragments, grouping operations for multiple fragments and redesigning the process definition user interface.
- The actions for making document modification more flexible are more difficult, as this requires redesigning the process XML file structure and adding more attributes into the Document XML file.
- The tool provides mechanisms to support best-practice guidelines for designing worked examples and faded worked examples. However, these guidelines are not directly pointed out to the teacher by the tool. Hence, it is worth creating a video to explain these guidelines and demonstrate how to build effective worked examples by applying them.

The advantage of improving the Teacher User Interface to be more user-friendly and intuitive is that it would reduce the learning curve for teachers starting to

use IWE and enable more teachers to use the tool to develop interactive worked examples or transfer existing static worked examples into interactive format.

Improving the Deployment of IWE

IWE was deployed as a PC-based application. Students were able to use it either in the lab or on their own computers. However, students had to download the system to their home machines; it was not possible to collect usage data for students working at home and collecting feedback and answers was also more difficult. Creating a web-based version of IWE for student use would overcome these problems in our environment and would also open up the opportunities for wider use of IWE.

8.3.2 Possible Further Research

8.3.2.1 Measuring the Educational Value of IWE

The first further research question is: Can we conduct a more wide-ranging experiment on the educational value of interactive worked examples in Computing Science by getting more teachers and different groups of students involved?

Based on strong evidence from the literature review, presented in Chapter 2, it has been assumed that worked examples are educationally beneficial, provided that best-practice approaches are followed in their construction and delivery, as was done in this research. However, there may be some specific advantages that using interactive worked examples brings that can be measured experimentally. So, education researchers could carry out experiments using IWE to measure whether using interactive worked examples enhances students' learning in the academic environment. There would also be the possibility of carrying out experiments in computer education to assess whether transformation-based worked examples do address any of the well-known problems in teaching computing science.

The two enhancements to IWE, discussed in section 8.3.1 would make these types of experiments more feasible. Firstly, creating a web interface for

students to explore interactive worked examples would make delivery of the worked examples to students easier and the collection of feedback and usage data simpler. Secondly, improvements to the Teacher User Interface to make it more user-friendly should encourage more teachers to develop interactive worked examples.

For example, deploying IWE in the high school context would give significant opportunity to evaluate large scale use of the tool. However, the current implementation, as a stand-alone Java program, is not suitable for school deployment. A development initially would be to build a web-based system for the delivery of interactive worked examples. The web interface would be available in any school classroom, without significant change to schools' computer systems, and would be able to capture usage data on a large scale. The current Java-based system for creating interactive worked examples could be extended to allow upload and download of the materials to/from the web-based system to enable teachers to create new and edit existing worked examples. The expectation is that a relatively small number of enthusiastic teachers would create new worked examples, but that any teacher might wish to adjust a worked example to better meet their own needs. The questioning and logging features of the current system could be transferred to the web-based system, and analyses of student behaviour, individually and collectively, presented back to teachers via a teacher interface, to directly inform their teaching. At the same time, this extended use by a large numbers of teachers and learners would provide excellent further data for analysing the usability of the tool and understanding more about the creation and deployment of interactive worked examples.

8.3.2.2 Using IWE in Other Subjects

The second further research question is: Can we find other academic subjects that have transformation problems for which IWE could be used to provide interactive worked examples?

The researcher is in the computing science area; so all the worked examples built belong to this area. It would be valuable to try other subjects, as

transformation-based problems do not only exist in computing science. Three possible topics are briefly outlined below.

Answering Essay-based Examination Questions

One possible scenario suggested by a colleague in Psychology is to demonstrate how to construct good answers to essay-based examination questions, as written critical essays are widely applicable to any subjects. The basic idea is:

1. Read the question carefully and then make a list of all the topics that can be remembered that might be relevant to the question.
2. Work through the list of topics and identify those that most relevant to the question.
3. Organise these topics into a logical order to give a structure for the essay and then write the essay by filling in the details.

This could be presented in IWE by using three document panes. The first pane would display the question. The second pane would then present a list of possible topics. The process would then step through this list identifying the relevant topics; for each topic an explanation would be given as to why it was included or excluded. The third pane would then build up the list of relevant topics in a logical order, explaining why the structure was chosen. It would also be possible to fade this type of worked example by asking students to explain why given topics were relevant or not relevant.

It is interesting to note that this approach is similar to a technique used in Object-Oriented Design. Given a problem description or requirements document a list of all nouns and noun phrases is created. From this list a list of possible classes is identified; other nouns may relate to attributes of classes; be a different name for a class already identified; or simply be irrelevant background information. This list of potential classes is then used to create an outline class diagram and associations between classes are identified. The process can be continued by adding attributes to the classes and then methods, which will be represented by verbs in the problem description. [113]

Improving Academic Writing Skills

Another example was found from a workshop on the topic of Teaching and Learning held in the University of Glasgow recently [114]. Figure 8.1 shows a worked example developed using PowerPoint, which was used as an online tutorial to explain the difference between informal and formal writing, and personal and impersonal writing, in order to promote skills relating to academic writing. Transferring these existing resources to IWE would not be a hard task.

The left-hand example shown in Figure 8.1 could be presented in IWE by using two document panes; one for informal text, and the other for formal text. Faded worked examples can be used to represent this example; possible steps in the process are described below.

1. Ask the student to read the informal text;
2. Ask the student to convert into formal text as a question;
3. Show the formal text of rewording this informal text;
4. Highlight the difference between these two documents, which is highlighted in red in Figure 8.1, with the teacher's explanation added.

The right-hand example of Figure 8.1 could be transferred in a similar way. A question could also be asked to identify the personal words in the personal text.

It would be interesting to conduct an experiment to compare the educational value of using examples created in PowerPoint, like the one shown in Figure 8.1, and the same worked examples created using IWE.

Figure 8.1 studySMART online Tutorials for Teaching Academic Writing[114]

Statistics

Statistics could be another possible subject to use IWE. According to Marriott et al. [115], a simple paradigm for solving problems using statistics can be summarised by using four activities, which is shown in Figure 8.2. The problem-solving process is a decision-making process, which can be connected with students' prior learning and experiences. For example, to specify the problem and plan: requires formulation of the questions in terms of the data needed and consideration of what inferences can be drawn from the data; deciding what data to collect (including sample size and data format); and what statistical analysis is needed.

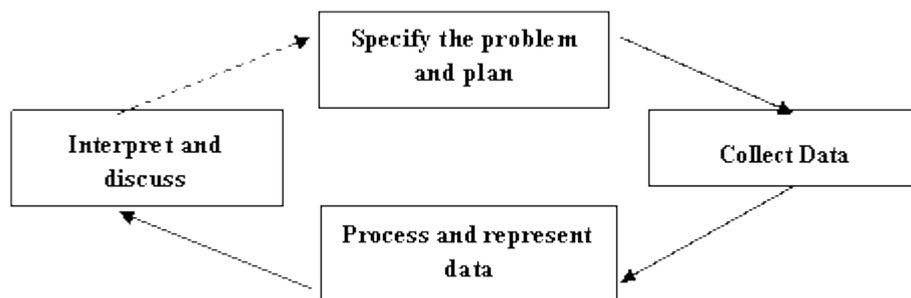


Figure 8.2 The Statistical Problem Solving Approach[40]

Statistics involves complex real-world problems, and it is (very) hard as a novice to know the right statistical methods to use to analyse the real-world situation. This process of identifying the right statistic methods is just like the computing science problems explored in this dissertation - in that a model of what is going on must be created, and the process of creating models is not algorithmic. When a statistics novice tries to solve the modelling problem from first principles, it is very hard indeed. But an expert just looks at the problem and immediately recognises how it matches to their extensive set of patterns of problems seen before, along with the appropriate set of statistical procedures to follow to solve the problem. They see the problem as an exemplar of one of a whole bunch of patterns they've developed over time, just as it has been argued that the computing science people need also to do.

8.3.2.3 Summary

The evaluation of the prototype of IWE has demonstrated that it is possible to create an authoring environment for building usable interactive worked

examples. There are interesting possibilities for future research, both within the computing science community and in other subjects, as outlined above.

However, to pursue these future research directions it is necessary to provide a web interface for students so that interactive worked examples can be made available to much a wider pool of student users. Improving the teacher user interface should encourage more teachers to get involved in creating interactive worked examples.

Bibliography

1. Cooper, R. and C. Wang. Implementing Immersive Learning Environments For Teaching Software Engineering. in 7th Annual Conference of the ICS HE Academy Trinity College, Dublin. 2006.
2. AquaFold, Aqua Data Studio, 2013: 440 N Wolfe Rd. Sunnyvale, CA 94085.
3. Polya, G., How to Solve It: A New Aspect of Mathematical Method 2004: Princeton University Press.
4. Gamma, E., et al., Design patterns: elements of reusable object-oriented software 1994: Pearson Education.
5. Renkl, A., The Worked-Out Examples Principle in Multimedia Learning, in Cambridge Handbook of Multimedia Learning, R.E. Mayer, Editor 2005 Cambridge University Press.
6. Mayer, R.E., Learning and Instruction. 2 ed 2007: Pearson Prentice Hall.
7. Renkl, A. and R. Atkinson, Fading from Worked Examples to Practice Problems: An Instructional Design Model, T.E. GUIDE, Editor 2006.
8. Murray, M.C. and M. Guimaraes, Animated database courseware: using animations to extend conceptual understanding of database concepts. J. Comput. Small Coll., 2008. 24(2): p. 144-150.
9. Mayes, J.T. and C.J. Fowler, Learning technology and usability: a framework for understanding courseware. Interacting with Computers, 1999. 11(5): p. 485-497.
10. Buckingham, S.S., Learning analytics, 2012, UNESCO Policy Brief.
11. Robins, A., J. Rountree, and N. Rountree, Learning and teaching programming: A review and discussion. Computer Science Education, 2003. 13(2): p. 137-172.
12. Connolly, T.M. and C.E. Begg, Database System_A Practical Approach to Design, Implementation, and Management. 4th ed 2005: Addison-Wesley.
13. Atkinson, R.C. and R.M. Shiffrin, Human Memory: A Proposed System and its Control Processes, in Psychology of Learning and Motivation, W.S. Kenneth and S. Janet Taylor, Editors. 1968, Academic Press. p. 89-195.
14. Sweller, J., Cognitive load during problem solving: Effects on learning. Cognitive Science, 1988. 12(2): p. 257-285.
15. Davies, M., Concept mapping, mind mapping and argument mapping: what are the differences and do they matter? Higher Education, 2011. 62(3): p. 279-301.
16. Sweller, J., Human Cognitive Architecture, in Handbook of research for educational communications and technology, Spector, M., et. al., Editors. 2008, Lawrence Erlbaum Associates. p. 369-383.
17. Cowan, N., Working Memory Capacity 2005: Psychology Press.
18. Renkl, A. and R.K. Atkinson, Structuring the Transition From Example Study to Problem Solving in Cognitive Skill Acquisition: A Cognitive Load Perspective. EDUCATIONAL PSYCHOLOGIST, 2003. 38(1): p. 15-22.
19. Ross, P.E., The expert mind. Scientific American, 2006. 295(2): p. 64-71.

Bibliography

20. Soloway, E. and K. Ehrlich, Empirical studies of programming knowledge. *Software Engineering, IEEE Transactions on*, 1984(5): p. 595-609.
21. Kirschner, P.A. and J. Van Merriënboer, *Ten Steps to Complex Learning A New Approach to Instruction and Instructional Design*. 2008.
22. Anderson, J.R., J.M. Fincham, and S. Douglass, The role of examples and rules in the acquisition of a cognitive skill. *Journal of experimental psychology. Learning, memory, and cognition*, 1997. **23**(4): p. 932-45.
23. Tuovinen, J.E. and J. Sweller, A comparison of cognitive load associated with discovery learning and worked examples. *Journal of Educational Psychology*, 1999. **91** (2): p. 334-341.
24. Paas, F., A. Renkl, and J. Sweller, Cognitive Load Theory and Instructional Design: Recent Developments. *EDUCATIONAL PSYCHOLOGIST*, 2003. **38**(1): p. 1-4.
25. Sweller, J., J. Van Merrienboer, and F. Paas, Cognitive Architecture and Instructional Design. *Educational Psychology Review*, 1998. **10**(3): p. 251-296.
26. Lave, J. and E. Wenger, *Situated Learning: Legitimate Peripheral Participation*, 1991, Cambridge University Press: Cambridge, England. p. 27-42.
27. Collins, A., J. Brown, and A. Holum, *Cognitive Apprenticeship: Making Thinking Visible*. *American Educator*, 1991. **6**(11): p. 38-46.
28. Collins, A., J.S. Brown, and S.E. Newman, Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics, in *Cognition and instruction: Issues and agendas*, L.B. Resnick, Editor 1989, Lawrence Erlbaum Associates. p. 453-494.
29. Atkinson, R.K., et al., Learning from Examples: Instructional Principles from the Worked Examples Research. *Review of Educational Research*, 2000. **70**(2): p. 181-214.
30. Kirschner, P.A., J. Sweller, and R.E. Clark, Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *EDUCATIONAL PSYCHOLOGIST*, 2006. **41**(2): p. 75-86.
31. Sweller, J., P.A. Kirschner, and R.E. Clark, Why minimally guided teaching techniques do not work: A reply to commentaries. *EDUCATIONAL PSYCHOLOGIST*, 2007. **42**(2): p. 115-121.
32. Laurillard, D., *Rethinking university teaching: a conversational framework for the effective use of learning technologies 2ed* 2002, London and New York: Routledge.
33. Kolb, D.A., *Experiential learning: experience as the source of learning and development* 1984, Englewood Cliffs, NJ: Prentice Hall.
34. Kalyuga, S., Expertise Reversal Effect and Its Implications for Learner-Tailored Instruction. *Educational Psychology Review*, 2007. **19**(4): p. 509-539.
35. Murray, M.C. and M. Guimaraes, Animated Courseware Support for Teaching Database Design. *Issues in Informing Science & Information Technology*, 2009(6): p. 201-211.

Bibliography

36. Arnow, D.M., :-)When you grade that: using e-mail and the network in programming courses, in Proceedings of the 1995 ACM symposium on Applied computing 1995, ACM: Nashville, Tennessee, United States. p. 10-13.
37. Petersen, A., M. Craig, and D. Zingaro, Reviewing CS1 exam question content, in Proceedings of the 42nd ACM technical symposium on Computer science education 2011, ACM: Dallas, TX, USA. p. 631-636.
38. Paas, F. and T. van Gog, Optimising worked example instruction: Different ways to increase germane cognitive load. Learning and Instruction, 2006. 16(2): p. 87-91.
39. Sweller, J. and G.A. Cooper, The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. Cognition and Instruction, 1985. 2(1): p. 59-89.
40. Trafton, J.G. and B.J. Reiser, The contributions of studying examples and solving problems to skill acquisition, 1994, Princeton University.
41. Wittwer, J. and A. Renkl, Why Instructional Explanations Often Do Not Work: A Framework for Understanding the Effectiveness of Instructional Explanations. EDUCATIONAL PSYCHOLOGIST, 2008. 43(1): p. 49-64.
42. Reed, S.K. and C.A. Bolstad, Use of examples and procedures in problem solving. Journal of Experimental Psychology: Learning, Memory, and Cognition, 1991. 17(4): p. 753.
43. Cooper, G.A. and J. Sweller, Effects of schema acquisition and rule automation on mathematical problem-solving transfer. Journal of Educational Psychology, 1987. 79(4): p. 347-362.
44. Chi, M.T.H., et al., Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems. Cognitive Science, 1989. 13(2): p. 145-182.
45. Chi, M.T.H., Self-explaining: The dual processes of generating inference and repairing mental models, in Advances in instructional psychology: Educational design and cognitive science, Vol. 5. 2000, Mahwah, NJ, US: Lawrence Erlbaum Associates Publishers. p. 161-238.
46. Renkl, A., Learning from worked-out examples: A study on individual differences. Cognitive Science, 1997. 21(1): p. 1-29.
47. Chi, M.T., Constructing self-explanations and scaffolded explanations in tutoring. Applied Cognitive Psychology, 1996. 10(7): p. 33-49.
48. Renkl, A., et al., Learning from worked-out examples: The effects of example variability and elicited self-explanations. Contemporary educational psychology, 1998. 23(1): p. 90-108.
49. Roy, M. and M.T. Chi, The self-explanation principle in multimedia learning. The Cambridge handbook of multimedia learning, 2005: p. 271-286.
50. Renkl, A., R.K. Atkinson, and U. Maier, H. , From studying examples to solving problems: Fading worked-out solution steps helps learning, 2000, Erlbaum.

Bibliography

51. FRIZE, M. and C. FRASSON, Decision-support and intelligent tutoring systems in medical education. *Clinical and investigative medicine*, 2000. **23**(4): p. 266-269.
52. Schwonke, R., et al. Can tutored problem solving benefit from faded worked-out examples? in *Proceedings of EuroCogSci 07*. 2007. New York: NY: Erlbaum.
53. Atkinson, R.K. and S.J. Derry, Computer-Based Examples Designed to Encourage Optimal Example Processing: A Study Examining the Impact of Sequentially Presented, Subgoal-Oriented Worked Examples. 2000.
54. Bloom, B.S., *Learning for mastery*, 1968, Regional Education Laboratory for the Carolinas and Virginia Durham, NC.
55. Hundhausen, C.D., S.A. Douglas, and J.T. Stasko, A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 2002. **13**(3): p. 259-290.
56. Murray, M.C. and et al. *Animated DataBase Courseware*. [cited 2012 01/03]; Available from: <http://adbc.kennesaw.edu/>.
57. Laurillard, D., *Productivity Achieving higher quality and more effective learning in affordable and acceptable ways*, 2011: London Knowledge Lab.
58. Valentine, D.W., CS educational research: a meta-analysis of SIGCSE technical symposium proceedings. *ACM SIGCSE Bulletin*, 2004. **36**(1): p. 255-259.
59. Pears, A., et al., A Survey of Literature on the Teaching of Introductory Programming. *SIGCSE Bull.*, 2007. **39**(4): p. 204-223.
60. Murray, M.C., Personal Communication: Re: Interesting in ADbC project, 2013.
61. Mishra, P. and M.J. Koehler, Technological Pedagogical Content Knowledge: A Framework for Teacher Knowledge. *Teachers College Record*, 2006. **108**(6): p. 1017-1054.
62. Shulman, L.S., *Those Who Understand: Knowledge Growth in Teaching*. *Educational Researcher*, 1986. **15**(2): p. 4-14.
63. Harris, J.P.M., Teachers' Technological Pedagogical Content Knowledge and Learning Activity Types: Curriculum-based Technology Integration Reframed. *Journal of Research on Technology in Education*, 2009. **41**(4): p. 393-416.
64. Archambault, L. and K. Crippen, Examining TPACK Among K-12 Online Distance Educators in the United States. *Contemporary Issues in Technology and Teacher Education*, 2009. **9**(1): p. 71-88.
65. Locatis, C. and H. Al-Nuaim, Interactive technology and authoring tools: A historical review and analysis. *Educational Technology Research and Development*, 1999. **47**(3): p. 63-75.
66. Bodendorf, F., M. Schertler, and E. Cohen, Producing Reusable Web-Based Multimedia Presentations. *Interdisciplinary Journal of Knowledge and Learning Objects*, 2005. **Volume 1**: p. 127-142.
67. Recker, M. and e. al., *Teaching, Designing, and Sharing: A Context for Learning Objects*. *Interdisciplinary Journal of Knowledge and Learning Objects*, 2005. **Volume 1**,: p. 197-216.

Bibliography

68. Hsiao, I.H., S. Sosnovsky, and P. Brusilovsky, Guiding students to the right questions: adaptive navigation support in an E-Learning system for Java programming. *Journal of Computer Assisted Learning*, 2010. **26**(4): p. 270-283.
69. Avner, A., S. Smith, and P. Tenczar, CBI authoring tools: Effects on productivity and quality. *Journal of Computer-Based Instruction*, 1984. **11**(3): p. 85-89.
70. Alevan, V., et al. The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. in *Intelligent Tutoring Systems*. 2006. Springer.
71. Ainsworth, S., Using a Single Authoring Environment across the Lifespan of Learning. *Educational Technology & Society*, 2007. **10**(3): p. 22-31.
72. Paquette, L., J.-F. Lebeau, and A. Mayers, Authoring Problem-Solving Tutors: A Comparison between ASTUS and CTAT, in *Advances in Intelligent Tutoring Systems*, 2010. Springer. p. 377-405.
73. Murray, T., Authoring Knowledge-Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. *Journal of the Learning Sciences*, 1998. **7**(1): p. 5-64.
74. Karavirta, V., et al., MatrixPro-A Tool for On-The-Fly Demonstration of Data Structures and Algorithms, in *IEEE International Conference on Advanced Learning Technologies*, 2004. p. 892-893.
75. Munro, A., et al., Authoring simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*, 1997. **8**(3-4): p. 284-316.
76. Van Joolingen, W.R. and T. De Jong, Simquest, in *Authoring tools for advanced technology learning environments*, 2003, Springer. p. 1-31.
77. Adobe Systems Software Ireland Ltd. Adobe Captivate. 2013 12-07-2014]; Available from: <http://www.adobe.com/uk/products/captivate.html>.
78. WebSoft Ltd. Course Lab. 1999 12-07-2014]; Available from: http://www.courselab.com/view_doc.html?mode=home.
79. Learning Technology Research Institute. GLO Maker. 2009 12-07-2014]; Available from: <http://www.glomaker.org/>.
80. Basuhail, A.A. Design and Implementation Of and E-learning Content Using Simple and Obtainable Tools. in *First International Conference 'E-learning and Distance Education'*. [Electronic version]. Retrieved. 2009.
81. Freitas, S.d. and T. Neumann, The use of 'exploratory learning' for supporting immersive learning in virtual environments. *Comput. Educ.*, 2009. **52**(2): p. 343-352.
82. JEUNG, H.-J., P. CHANDLER, and J. SWELLER, The role of visual indicators in dual sensory mode instruction. *Educational psychology*, 1997. **17**(3): p. 329-343.
83. ISO. Usability Defination. [cited 2014 28/03/2014]; Available from: http://www.usabilitynet.org/management/b_what.htm.
84. Nielsen, J., *Usability Engineering* 1994: Elsevier.

Bibliography

85. Dumas, J.S. and J. Redish, A practical guide to usability testing 1999: Intellect Books.
86. Preece, J. and D. Benyon, A Guide to Usability: Human Factors in Computing 1993: Addison-Wesley Longman Publishing Co., Inc.
87. Albert, W. and D. Tedesco, Reliability of self-reported awareness measures based on eye tracking. *Journal of usability studies*, 2010. 5(2): p. 50-64.
88. Kirakowski, J. and M. Corbett, SUMI: the Software Usability Measurement Inventory. *British Journal of Educational Technology*, 1993. 24(3): p. 210-212.
89. Lewis, J.R., Psychometric Evaluation of the Post-Study System Usability Questionnaire: The PSSUQ. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 1992. 36(16): p. 1259-1260.
90. Harper, B.D. and K.L. Norman. Improving user satisfaction: The questionnaire for user interaction satisfaction version 5.5. in *Proceedings of the 1st Annual Mid-Atlantic Human Factors Conference*. 1993.
91. Brooke, J., SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 1996. 189: p. 194.
92. Brooke, J., SUS: A Retrospective. *Journal of usability studies*, 2013. 8(2).
93. Lewis, J.R. and J. Sauro, The factor structure of the system usability scale, in *Human Centered Design 2009*, Springer. p. 94-103.
94. Borsci, S., S. Federici, and M. Lauriola, On the dimensionality of the System Usability Scale: a test of alternative measurement models. *Cognitive processing*, 2009. 10(3): p. 193-197.
95. Tullis, T.S. and J.N. Stetson. A comparison of questionnaires for assessing website usability. in *Usability Professional Association Conference*. 2004.
96. Bangor, A., P.T. Kortum, and J.T. Miller, An empirical evaluation of the system usability scale. *Intl. Journal of Human-Computer Interaction*, 2008. 24(6): p. 574-594.
97. Sauro, J., A practical guide to the system usability scale: Background, benchmarks & best practices 2011: CreateSpace Independent Publishing Platform.
98. Rubin, J. and D. Chisnell, *Handbook of usability testing: how to plan, design, and conduct effective tests* 2008: John Wiley & Sons.
99. Squires, D. and J. Preece, Usability and learning: evaluating the potential of educational software. *Computers & Education*, 1996. 27(1): p. 15-22.
100. Squires, D. and J. Preece, Predicting quality in educational software: Evaluating for learning, usability and the synergy between them. *Interacting with Computers*, 1999. 11(5): p. 467-483.
101. Hornbæk, K., Current practice in measuring usability: Challenges to usability studies and research. *International journal of human-computer studies*, 2006. 64(2): p. 79-102.
102. Avouris, N.M., et al., Evaluation of distance-learning environments: Impact of usability on student performance. *International Journal of Educational Telecommunications*, 2001. 7(4): p. 355-378.

Bibliography

103. Tselios, N., N. Avouris, and V. Komis, The effective combination of hybrid usability methods in evaluating educational applications of ICT: Issues and challenges. *Education and Information Technologies*, 2008. **13**(1): p. 55-76.
104. AquaFold. Aqua Data Studio 2006 [cited 2014 12/12]; Available from: <http://www.aquafold.com/aquadatastudio.html>.
105. Hollender, N., et al., Integrating cognitive load theory and concepts of human-computer interaction. *Computers in Human Behavior*, 2010. **26**(6): p. 1278-1288.
106. Apache. Log4J. [cited 2015 20/01]; Available from: <http://logging.apache.org/log4j/2.x/>.
107. NNL_Technology_AB. InfoNode Docking Windows. [cited 2013 06/06]; Available from: <http://www.infonode.net/>.
108. Tversky, B., J.B. Morrison, and M. Betrancourt, Animation: can it facilitate? *International journal of human-computer studies*, 2002. **57**(4): p. 247-262.
109. Mishra, J. and A. Mohanty, *Software Engineering*. 1st ed 2011: Pearson; .
110. SurveyMonkey. [cited 2014 20/03]; Available from: <http://www.surveymonkey.com>
111. Dann, W.P., S. Cooper, and R. Pausch, *Learning to Program with Alice (w/CD ROM)* 2011: Prentice Hall Press.
112. Ben-Bassat Levy, R., M. Ben-Ari, and P.A. Uronen, The Jeliot 2000 program animation system. *Computers & Education*, 2003. **40**(1): p. 1-15.
113. Booch, G., et al., *Object-oriented analysis and design with applications*. Vol. 3.
114. McKay, J. and D. McKendrick, Bridging the gap: Supporting the FE/HE transition using multimedia and social learning tools, in *University of Glasgow Alumin Teaching and Learning Events 2013: Learning and Teaching Centre*, University of Glasgow.
115. Marriott, J., N. Davies, and L. Gibson, Teaching, learning and assessing statistical problem solving. *Journal of Statistics Education*, 2009. **17**(1): p. 1.

Appendices

Appendix 1: Participant Consent Form: IWE Usability Survey

Title of Project: An Authoring Environment for Creating Interactive Worked Examples

Name of Researcher: Yulun Song

The aim of this experiment is to investigate the usability of the IWE for students. The experiment will take about 40 minutes to complete and the whole process of this survey will be recorded by video under your permission.

This survey will be run under the condition of one-to-one and side by side observation. You will be guided through the initial stages of IWE to a point where you are ready to start running a worked example.

During the experiment, you will be asked to carry out several tasks. During the performance, you are welcomed and encouraged to ask questions or make comments, meanwhile, some questions will be asked by the observer. Finally you will be asked to fill in a short questionnaire and will be given a short interview.

All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Video data will be stored in a password protected computer account; paper data will be kept in a lock draw in a locked office.

Your participation in this experiment is voluntary and will have no effect on your marks for any subject at this, or any other university. You are free to withdraw at any time, without giving any reason.

Please note that it is the teaching tool which is a piece of software, not you, which are being evaluated. You are free to withdraw from the experiment at any time without giving any reason, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:

Yulun Song

School of Computing Science

Room F151, 18 Lilybank Gardens, Glasgow

yulun@dcs.gla.ac.uk

I confirm that I have read this information sheet. I understand the Plain Language Statement for the above study and have had the opportunity to ask questions. Hence, I agree to voluntarily take part in this experiment:

Name: _____ Email: _____

Signature: _____ Date: _____

If you would like to receive an overview of the results obtained in this experiment, please tick here

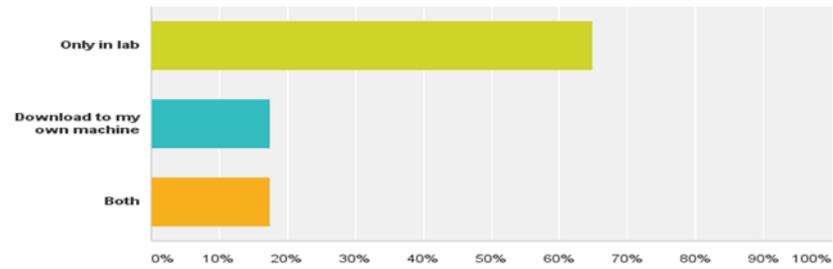
This study adheres to the BPS ethical guidelines, and has been approved by the FIMS ethics committee of The University of Glasgow (ref: CSE01135). Whilst you are free to discuss your participation in this study with the researcher (contactable on 0141 330 8573), if you would like to speak to someone not involved in the study, you may contact the chair of the FIMS Ethics Committee (<http://ethics.ims.gla.ac.uk/>).

Appendix 2: IWE Evaluation Questionnaire

1. Where did you use IWE?
Only in lab Download to my own machine Both
2. How easy do you think it is to use the tool?
Very Easy Easy Neutral Hard Very Hard
3. Do you think you can enhance your understanding through the use of IWE?
Yes Not Sure No
4. Do you think worked examples in IWE were easy to follow compared with worked examples in a text book or reading through PowerPoint slides when working on you own?
Much Easier Easier Same Harder Much Harder
5. Did you send any feedback messages or questions to your lecturer, after interacting with any of the worked examples?
Yes No
If "No", what are the reasons that stopped you sending messages back to your lecturer?
 - I wanted to, but I did not know how to, send my message.
 - The worked examples explained everything very clearly, I understood everything.
 - I did not believe the lecturer would receive my messages and reply.
 - I still prefer face-to-face talking rather than using this method.
 - The tool is hard to use to write messages and send feedback.
 - It takes time to type a feedback message.
6. The feedback function of IWE is useless.
Strongly Agree Agree Neutral Disagree Strongly Disagree
7. I was very confused about the feedback mechanism. For example, should I ask questions about the worked examples, or should I write comments about the tool, or both.
Strongly Agree Agree Neutral Disagree Strongly Disagree
8. If you run through a complete database worked example for CS1Q, did you then select content in one panel, and see that the related content in the other panel was highlighted automatically? For example, selecting an attribute of an entity in an ER diagram, the related column in the table is highlighted.
Yes No
If 'Yes', what do you think of this correspondence function in this teaching tool?
Very Useful Useful Neutral Not Useful Not Useful at all
9. Did you experience any problems when using the tool, for example, finding worked examples, controlling the playing, or writing questions and sending them back to the lecturer?
Yes No
If 'Yes', please write down the problems you experienced.
10. If you have any other suggestions, please write them down. Thank you very much!

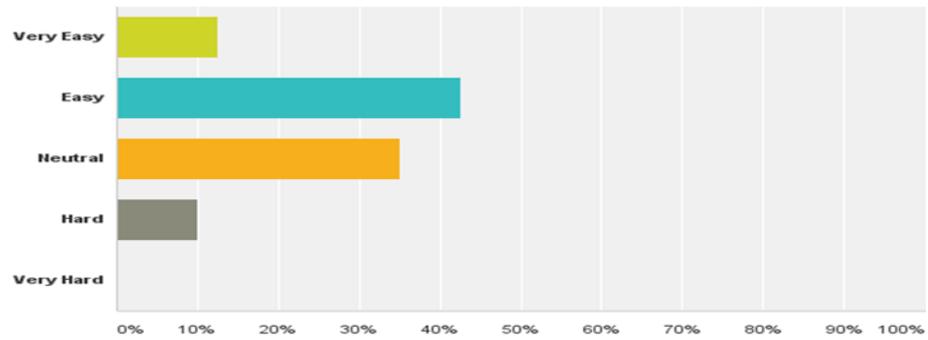
Appendix 3: Results of IWE Evaluation

Q1: Where did you use IWE?



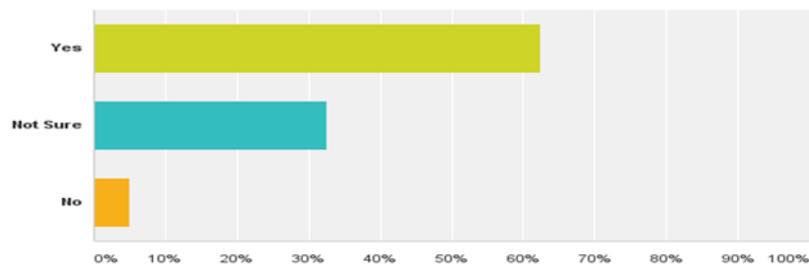
Answer Choices	Responses
Only in lab	65.00% 26
Download to my own machine	17.50% 7
Both	17.50% 7
Total	40

Q2: How easy do you think it is to use the tool?



Answer Choices	Responses
Very Easy	12.50% 5
Easy	42.50% 17
Neutral	35.00% 14
Hard	10.00% 4
Very Hard	0.00% 0
Total	40

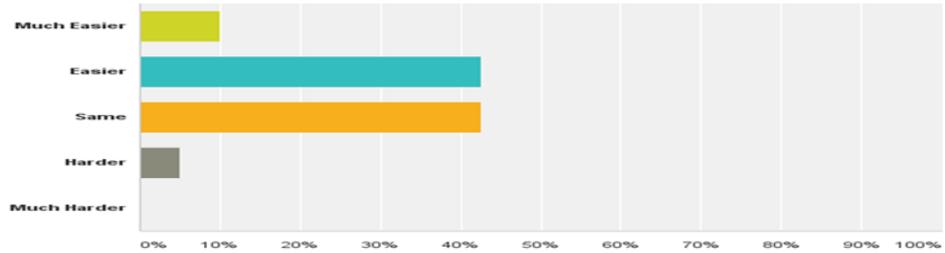
Q3: Do you think you can enhance your understanding through the use of IWE?



Answer Choices	Responses
Yes	62.50% 25
Not Sure	32.50% 13
No	5.00% 2
Total	40

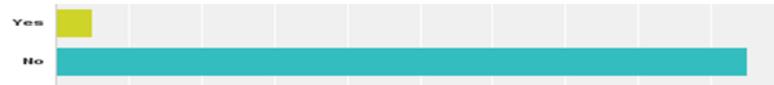
Appendix 3

Q4: Do you think worked examples in IWE were easy to follow compared with worked examples in a text book or reading through PowerPoint slides when working on you own?

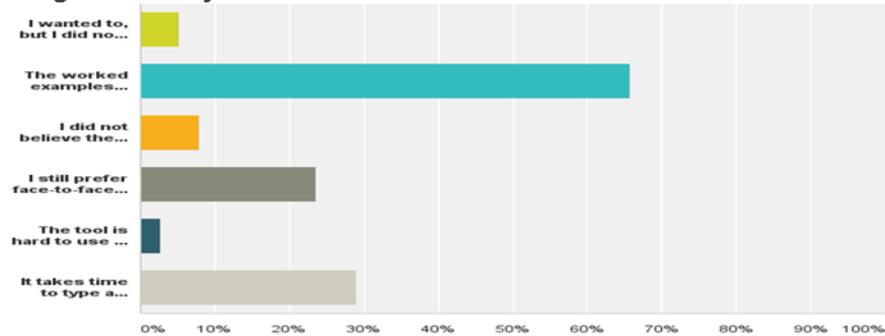


Answer Choices	Responses	Count
Much Easier	10.00%	4
Easier	42.50%	17
Same	42.50%	17
Harder	5.00%	2
Much Harder	0.00%	0
Total		40

Q5: Did you send any feedback messages or questions to your lecturer, after interacting with any of the worked examples?

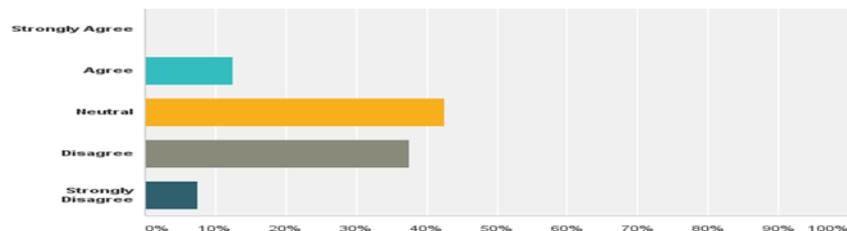


Sub_Q6: If choose No, What are the reasons that stopped you sending messages back to your lecturer?



Answer Choices	Responses	Count
I wanted to, but I did not know how to, send my message.	5.26%	2
The worked examples explained everything very clearly, I understood everything.	65.79%	25
I did not believe the lecturer would receive my messages and reply.	7.89%	3
I still prefer face-to-face talking rather than using this method.	23.68%	9
The tool is hard to use to write messages and send feedback.	2.63%	1
It takes time to type a feedback message.	28.95%	11
Total Respondents: 38		

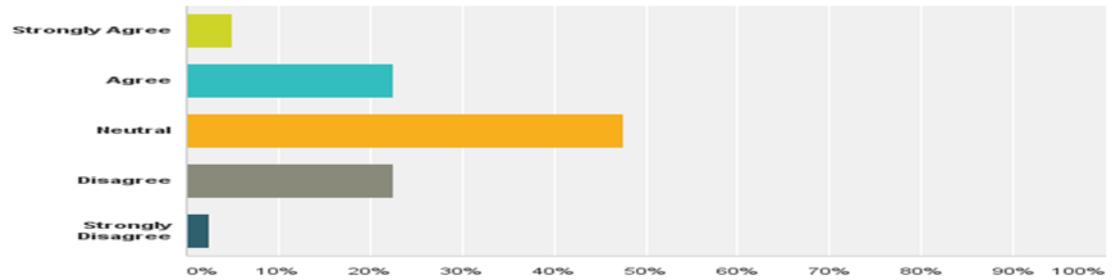
Q6: The feedback function of IWE is useless.



Answer Choices	Responses	Count
Strongly Agree	0.00%	0
Agree	12.50%	5
Neutral	42.50%	17
Disagree	37.50%	15
Strongly Disagree	7.50%	3
Total		40

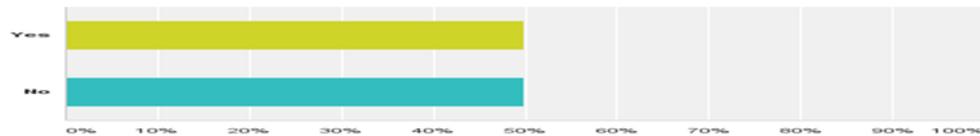
Appendix 3

Q7: I was very confused about the feedback mechanism. For example, should I ask questions about the worked examples, or should I write comments about the tool, or both.

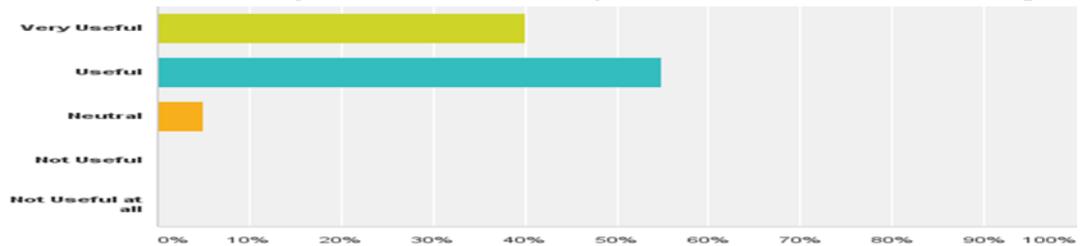


Answer Choices	Responses	Count
Strongly Agree	5.00%	2
Agree	22.50%	9
Neutral	47.50%	19
Disagree	22.50%	9
Strongly Disagree	2.50%	1
Total		40

Q8: If you run through a complete database worked example for CS1Q, did you then select content in one panel, and see that the related content in the other panel was highlighted automatically? For example, selecting an attribute of an entity in an ER diagram, the related column in the table is highlighted.



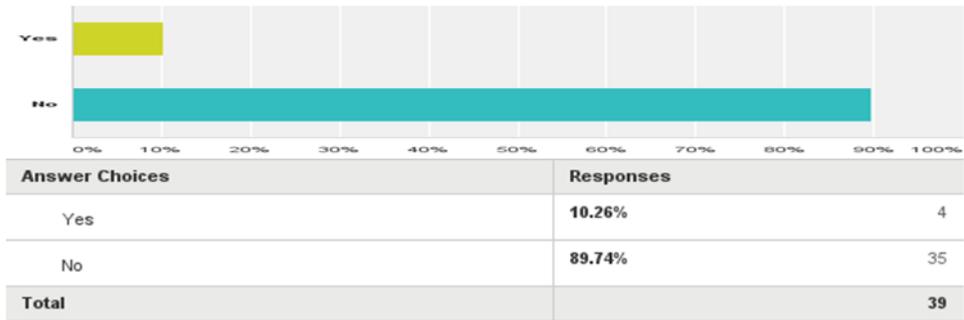
Sub-Q9: If 'Yes', what do you think of this correspondence function in this teaching tool?



Answer Choices	Responses	Count
Very Useful	40.00%	8
Useful	55.00%	11
Neutral	5.00%	1
Not Useful	0.00%	0
Not Useful at all	0.00%	0
Total		20

Appendix 3

Q9: Did you experience any problems when using the tool, for example, finding worked examples, controlling the playing, or writing questions and sending them back to the lecturer?



Q10: If you have any other suggestions, please write them down. Thank you very much!

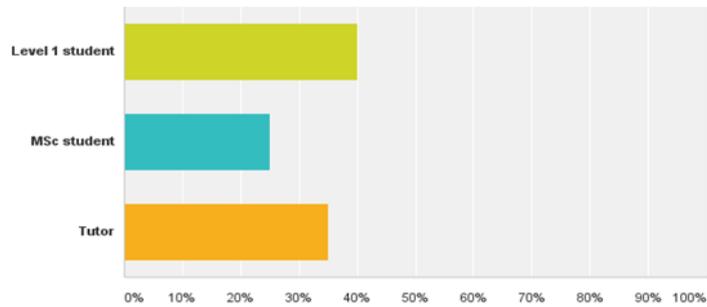
1. I only use CS1P examples.
2. Only used for CS1P part of program. Did not try CS1Q part, so missing some functions.
3. Only see the examples in CS1P. The Pattern example helped me better understand the contents which I did not understand during the lecture. Quite Useful.
4. Have examples for systems and more for programming.
5. I only used it once ever. So I have no opinion.
6. The "Your Answer" box looked like an edit box to me. I kept trying to type my answer in it. I think it should be more clearly marked as a read only widget. The step-by-step "Next" mode is preferable to the auto play mode.
7. The font style or size in the explanation area can be improved. (i.e. more readable at first sight)
8. Is it possible making this as a web application?
9. I find the lecturer's notes are easier to read.
10. The interface can be improved; I did not like that example for the lab exam practice. I had to resize one of the windows at the middle. All other features seem to be ok. I prefer face-to-face talk with the lecturer if possible. If I do not understand something or the lecture notes, I can ask directly after the lecture.
11. I did not have any problems with the courses learning, so I did not need this program very much.
12. I think that more online examples + proactive guesses would be more useful for my understanding of the subject.
13. Better than powerpointer slides or textbook examples as interactive and you can see how problems are solved.
14. Brief tutorial on how to best use the tool could be useful to make the most of it.

Appendix 4: One-to-one Usability Investigation Questions

1. Which level are you in?
Level 1 student MSc student Tutor
2. Did you think you finished all the tasks successfully?
Yes Not Sure No
3. When you played the worked example, did you look at all the changes in different panels?
Yes Not Sure No
4. When you answered a question, did you notice that your answer was recorded on the top right side panel?
Yes No
5. Which mode of looking at worked examples did you prefer?
 - Auto mode (using the Play button)
 - Manual mode (using the Next and Previous buttons)
 - Auto and manual combined.
 - I prefer auto play the example first, and then manually control it as required.
 - I prefer manually control the example first, and then auto play it again if needed.
6. How user-friendly is our software's interface?
 - Extremely user-friendly
 - Very user-friendly
 - Moderately user-friendly
 - Slightly user-friendly
 - Not at all user-friendly
7. How easy do you think to run a particular worked example by using the teaching tool?
Very Easy Easy Neutral Hard Very Hard
8. How easy do you think to find the correspondence function of this teaching tool?
Very Easy Easy Neutral Hard Very Hard
9. How easy do you think to find the help function of this teaching tool?
Very Easy Easy Neutral Hard Very Hard
10. Do you agree you can use this tool to run all the functions without any help?
Strongly Agree Agree Neutral Disagree Strongly Disagree
11. Do you agree you can use this tool to run all the functions with the help function of the system?
Strongly Agree Agree Neutral Disagree Strongly Disagree
12. Any other suggestions of the tool, please write them down. Thank you very much!

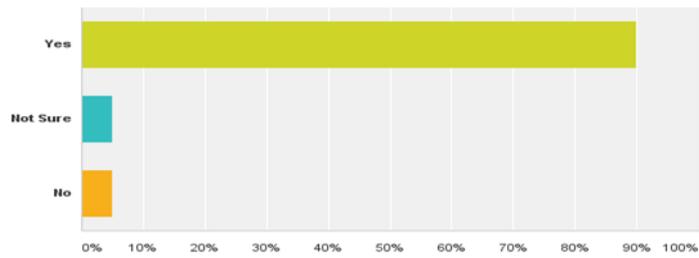
Appendix 5: Result of One-to-one Usability Investigation Survey

Q1: Which level are you in?



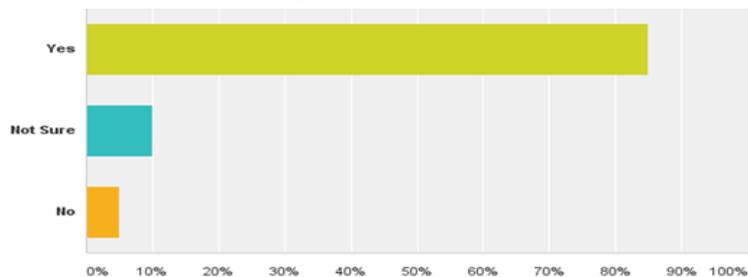
Answer Choices	Responses	Count
Level 1 student	40.00%	8
MSc student	25.00%	5
Tutor	35.00%	7
Total		20

Q2: Did you think you finished all the tasks successfully?



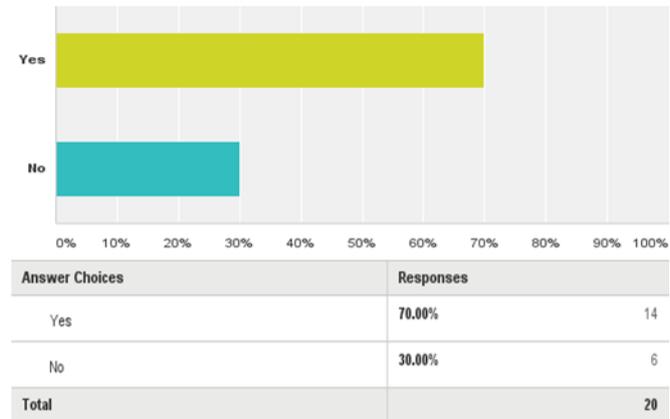
Answer Choices	Responses	Count
Yes	90.00%	18
Not Sure	5.00%	1
No	5.00%	1
Total		20

Q3: When you played the worked example, did you look at all the changes in different panels?

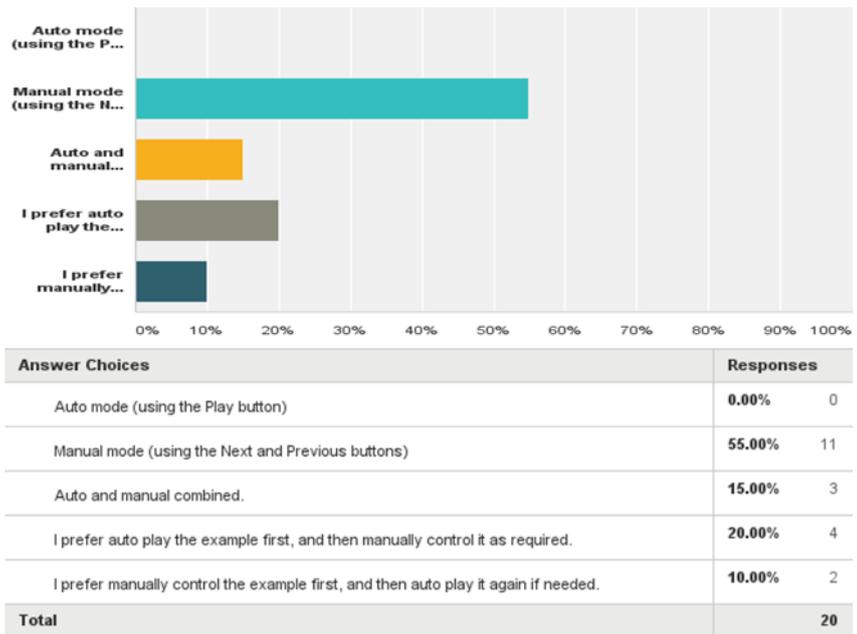


Answer Choices	Responses	Count
Yes	85.00%	17
Not Sure	10.00%	2
No	5.00%	1
Total		20

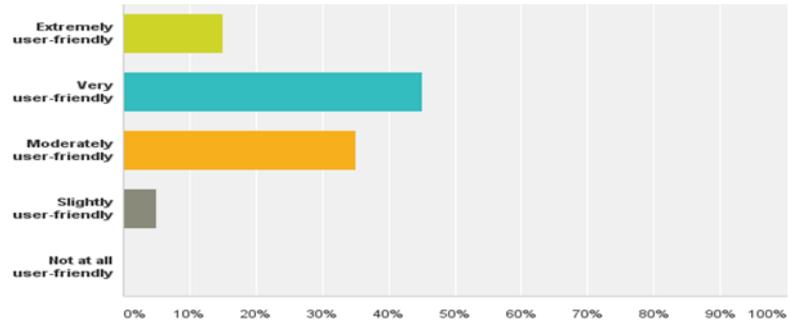
Q4: When you answered a question, did you notice that your answer was recorded on the top right side panel?



Q5: Which mode of looking at worked examples did you prefer?

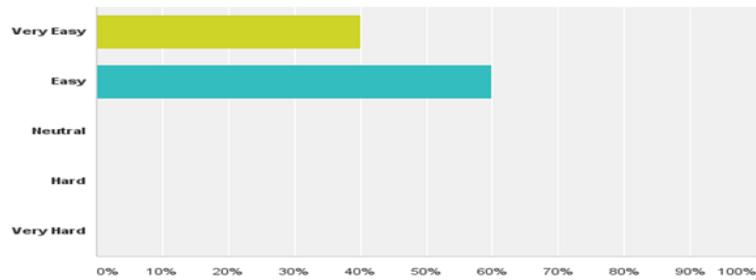


Q6: How user-friendly is our software's interface?



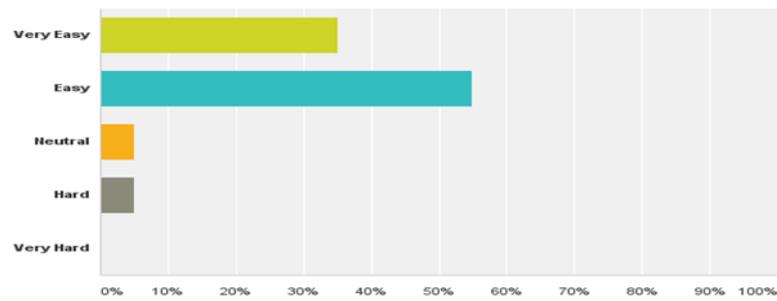
Answer Choices	Responses
Extremely user-friendly	15.00% 3
Very user-friendly	45.00% 9
Moderately user-friendly	35.00% 7
Slightly user-friendly	5.00% 1
Not at all user-friendly	0.00% 0
Total	20

Q7: How easy do you think to run a particular worked example by using the teaching tool?



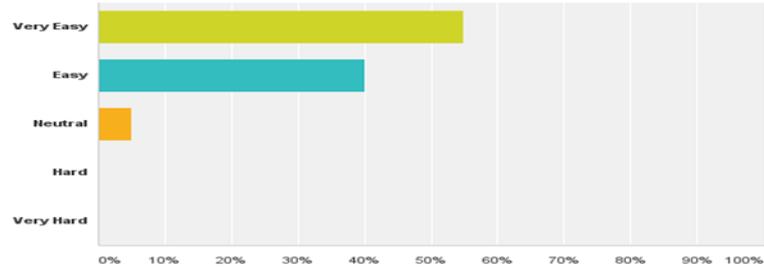
Answer Choices	Responses
Very Easy	40.00% 8
Easy	60.00% 12
Neutral	0.00% 0
Hard	0.00% 0
Very Hard	0.00% 0
Total	20

Q8: How easy do you think to find the correspondence function of this teaching tool?



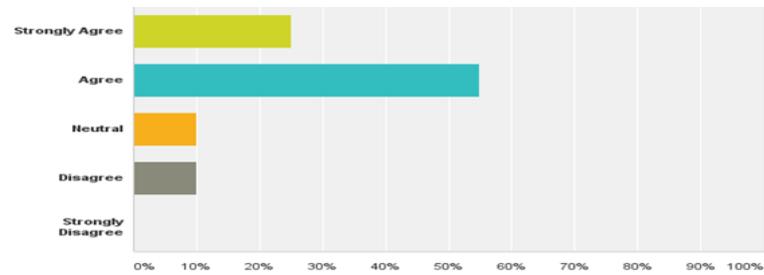
Answer Choices	Responses
Very Easy	35.00% 7
Easy	55.00% 11
Neutral	5.00% 1
Hard	5.00% 1
Very Hard	0.00% 0
Total	20

Q9: How easy do you think to find the help function of this teaching tool?



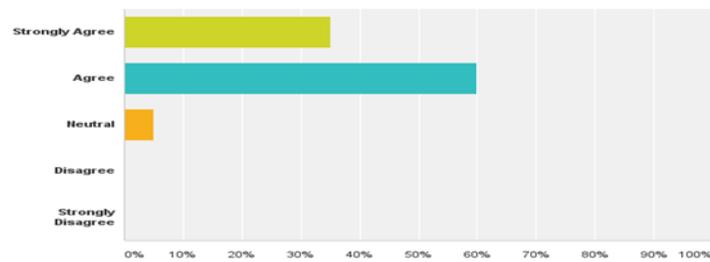
Answer Choices	Responses	Count
Very Easy	55.00%	11
Easy	40.00%	8
Neutral	5.00%	1
Hard	0.00%	0
Very Hard	0.00%	0
Total		20

Q10: Do you agree you can use this tool to run all the functions without any help?



Answer Choices	Responses	Count
Strongly Agree	25.00%	5
Agree	55.00%	11
Neutral	10.00%	2
Disagree	10.00%	2
Strongly Disagree	0.00%	0
Total		20

Q11: Do you agree you can use this tool to run all the functions with the help function of the system?



Answer Choices	Responses	Count
Strongly Agree	35.00%	7
Agree	60.00%	12
Neutral	5.00%	1
Disagree	0.00%	0
Strongly Disagree	0.00%	0
Total		20

Q12: Any other suggestions of the tool, please write them down. Thank you very much!

The other suggestions for the tool came out in interviews with subjects and are given in Appendix 6.

Appendix 6: Interview Results of One-to-one Usability Investigation

Interview Summary and Suggestions given by Usability Test Subjects between 28/01/2013 and 19/02/2013

Tutor's Suggestions

1. The subject is a tutor who taught both CS1P and CS1Q courses. She used the auto play mode to watch the CinemaExample, however, she changed the playing speed with longer waiting time. During the watching, she did not notice that her input answer for the questions was recorded and asked where the suggested answer can be found. She did not know the about the purpose of Feedback panel, however, she can adjust the "Scenario", "Answer" and "Feedback" panel size. Her eyes always focused on explanations first, and then looked at other highlighted content, depend on situation.

She suggested that in the Manual Control mode, after clicking the "Cancel" button in the popup dialog box, the step should directly go to next step.

In CS1Q Build ER example, the labels for relationship (1, M, N) are too small to see. Because she did not see the step 31 (which is the final step of the example) explanations carefully, the glossary tip and the correspondence function were ignored. Due to this, she suggested, in the explanation view, it should use different colour or font to emphasize it.

She suggested that if the lecturer can demo this tool inside the lecture, it would be much helpful for students using it after the lecture. More students will try the tool and the examples in their spare time, due to they were introduced the feature of the tool. In the lab, the priority is to finish the assessment and other homework, so the time for students using the tool and watching the examples is very limited. It only happened after they finished their must-done tasks.

2. The subject is a tutor who taught CS1Q course. She used the auto play mode to watch the BuildER example; however, sometimes she had to press the "Pause" button to stop to watch the content text. She did not notice the change in the answer view after responding a question. She had no idea about the purpose of the feedback view; due to the area did not look like to write any questions or comments.

She suggested that after running to the end of the worked example, popping up a dialog box to ask whether got any questions and pointing out where to write them down. The phrase of "Your Feedback" in the feedback view confused her as well. The GUI for inputting feedback and the feedback view needs to be improved.

She mentioned that the "Multi-value attribute" requires creating a separate table when transferring the ER diagram design into tables. If the students can see the examples before they dropped into the lab, she believed that the number of students who asked this question would be fewer.

Another popular problem is how to use Microsoft Access. She suggested it was better to show the table text like a real table in Microsoft Access.

She also pointed out that preferring to use the manual control mode, if in the real learning scenario for the first time running the example. For reviewing the example again, she would use the auto play mode, because it is more efficient and convenient.

Her eyes always focused on explanations first, and then looked at graphical content and then textual document.

7. The subject is a tutor who taught CS1Q course and used the Warm Up Demo and BuildER example to test the tool. He used the auto play mode with longer waiting time for each step to watch the Warm Up Demo example. When the answer dialog box popped out, he did not see the question at first. After replying the question, he did not notice his answer was recorded in the answer view. Then he run this example again using manual control mode and tried the correspondence and glossary functions. He clicked "Person" entity in ER diagram, but did not notice the "Person" in SQL command was highlighted automatically.

Running the BuildER example, he used the auto play mode first. However, the speed for playing first several steps was too fast to read all the textual description until came to the first question. He wanted to go back to see the problem description again and did not know what to do. After asking for help, he gave up answering the first question and decided to use manual control mode for the rest parts of example.

Appendix 6

Overview of the usage, he said: “when there was a question came out, I did think about the question very seriously and carefully. In the coming step, I compared my answer in the answer view (also in my mind) with the teacher’s model answer in the explanation view.”

He noticed the feedback view, but not sure the purpose of this part in the main GUI. Is it used to send feedback messages about the tool or send the feedback messages about the worked examples? If the later one, the messages would sent to whom? Who would response these messages and how can I receive responses of my questions?

He also mentioned that the correspondence function was more helpful for the bigger size of the example. Glossary and tutorial functions were good and easy to find, but it required some tips to tell students glossaries or tutorials were provided with this example as well.

His eyes focused on graphical document changes first, and then the highlight texts in yellow colour, finally the explanation texts.

He suggested that if the explanation texts can be sounded, which would be much better. If this example can be delivered earlier to the CS1Q course, it could definitely help students learning and understanding the way of designing ER diagram based on the textual description.

8. The subject is a tutor who taught CS1Q course and used the Warm Up Demo and BuildER example to test the tool. He used the manual control mode directly from beginning to the end. Although he saw the “Play” button, he still preferred to control everything under his control. After testing these two examples, he tested DBRelation example as well.

He run the Warm Up Demo example and tried correspondence function at the end. For “Person” entity, he did not notice the “Person” in SQL commands was highlighted. However, he noticed that the “Name Varchar (255)” was highlighted, when clicking “Name” attribute in ER diagram.

The glossary function was found with tips, due to he found the tooltip was shown on the graphical component when the mouse hover was on top of it.

He felt that the highlight colour was not strong enough and some colours confuse him. He did think about each question very carefully and seriously and read each step explanation very carefully. He also noticed his answer was recorded in the answer view and compared it with the teacher’s model answer.

As told the BuildER example was a bigger example, he minimized the scenario view and adjusted the size of answer view and feedback view in order to get more space for the watching space in the middle.

His eyes focused on highlighted text in problem description first; and then looked at the change in graphical document; finally watched the explanation texts.

He thought the amount of the questions in BuildER was OK and covered most of the questions asked during the tutorial and lab. During the demonstration, he tried to remind himself the definition of the relationship. So he used the glossary function as expected.

He believed that the correspondence function was extremely useful for level-1 students, especially, useful for doing the transformation from ER design to tables. Because the foreign key was missing in the ER design and he had to explain this again and again to each individual student how to add foreign keys into tables. He also suggested it was better to give more explanations on why adding attributes to a relationship.

When reviewing the example, he tried to compare his recorded answer with the teacher’s model answer. It was better to show them together.

The tutorial and help functions were found very quickly.

He said the best thing of this tool was the control of learning pace and time under the user’s expectation. If delivered early for CS1Q course, it would be very useful for students and tutors. It could definitely reduce some tutoring work amount.

The glossary, tutorial and the feedback functions are all very good feature to be integrated in this teaching tool. However, the way for students’ asking questions or writing comments could be more interactive as well.

11. The subject is a tutor who taught CS1P course and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he tested both auto play mode and manual control mode and mentioned preferring using the manual control mode and familiar with the GUI. He actively found the correspondence function and noticed his answer was recorded in the answer view after inputting.

Appendix 6

When running CinemaExample example, he used manual control mode, read the problem description and the initial explanation very carefully at the beginning. When there was a question, he thought about it very carefully and sometimes used draft paper to do some calculation. He compared his answer with module answer in the coming step carefully as well. When looking at the first question, he would like to go back to see the problem description again, however, the system did not allow doing this.

His eyes always focused on explanations first, and then looked at the highlighted text. If the highlighting happened in both plan document and code document, he looked at the plan first and then the code. He explained that this could help him to make a structure of the solution in the mind first, and then mapped how the solution can be related to the plan.

He guessed the purpose of feedback function was wrong. When he looked at the help content, he found how to use the feedback function properly.

He suggested that the text box for inputting answer was too small. It may be better to allow the user to click on a piece of code in the panel and then jumped back to that step, which this piece of code first added in. When reviewing the example again, it was better to allow the user jumping between steps as expected instead of clicking “Next” or “Previous” buttons several times. If possible, set up the correspondences in CinemaExample.

Glossary, tutorial and help functions were easy to be found. However, it was better to show glossary and tutorial in the top level in the menu bar, which beside to help menu. This would make these two functions even easy to be found.

15. The subject is a tutor who taught CS1P course and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he used manual control mode. Because he explained that the time for watching each step was not equal, using “Next” button to control would be more convenience. He thought the question very carefully. He did not find the correspondence function, although it had been explained. He though the automatically highlighting when clicking “Next” button was the correspondence function. After pointing out the operation, he realized this function was so useful and found the glossary function as well.

He mentioned the correspondence function might bring a little missing understanding or confusing for this type of demonstration. For example, if the attribute was deleted in the ER diagram, how about the related SQL commands? Do they automatic deleted by the system automatically, or these pieces of commands had to be removed by hand?

When running CinemaExamp, he used manual control mode and thought every question very carefully and seriously. He compared his answer with the model answer as well.

His eyes focused on explanation texts first, then the highlighting texts in the plan, finally looking at something other highlighting texts at most of the time. Sometimes because the plan can be remembered, might look at code before watching the plan. But read the explanation was always first priority.

He found the tutorial function was hard to find and the help was easy to find.

He guessed the purpose of feedback function wrong, after explaining he tested it. He picked the wrong starting point, which clicking the “Send Feedback” button first. After explain how to use it right, he confused with the index number in the feedback view, because he tried to link the step number with the index number. For example, got a question in step 20, writing the related question in the column with index number 20.

He suggested that the scenario view could be minimized automatically when playing a worked example. This could bring more space for looking the contents.

20. The subject is a tutor who taught CS1P course and used the Warm Up Demo and the CinemaExample examples to test the tool.

When running Warm Up Demo example, she used manual control mode. She noticed her answer was recorded in the answer view. When came to step 12, she found a little spelling mistake in the explanation, and wrote it directly into the feedback view. However, during the watching procedure, she pressed twice “Begin” button by mistake instead of pressing “Previous” button. She had to press “Next” button so many times in order to jump back to the original step. Hence, she suggested it was better allow the user go to a particular step directly, after the user had seen those steps. She actively found the correspondence function during the demonstration and the highlight colour confused her. For example, in a step the related contents were already highlighted, it was hard to

Appendix 6

know the new highlighting was brought through the interactive operation. So she suggested that it might only allow the user interactive with the content, after running all the steps.

She reviewed the example again and tested the auto play mode. The glossary function was found by providing tips, after finishing the review.

When running CinemaExamp, she used manual control mode. She thought and answered the very carefully. When coming to step 4 and 5, she felt so excited, because these two steps were exactly as same as she did in the lab tutoring students.

Her eyes always focused on explanation first, and then depending on the suggestion in the explanation. If the highlighting happened in both plan texts and problems, she would see the plan first. If the highlighting happened in plan texts and code texts, some time she would see the highlighted code, which was just added in, first without paying too much attention; and then looked at the explanation; finally looked at other highlighting contents.

She followed the suggestion in step 29 which asked to adjust the panel size for the problem description. In the final step, she adjusted the explanation view size as well.

She found the tutorial and helps functions very quickly.

She was so glad to use this tool, due to this tool can help her to explain some concepts which often quite hard to explain by using language. She said, if students using this tool and looked these kind of worked examples before they dropped into lab, the work amount of her would be reduced.

She also watched the Programming Pattern example, and commented that this kind of worked example integrated previous knowledge, and demonstrated the starting point of using your knowledge to solve problem. It was not only focused on introduce the concept, but also demonstrate the procedure of solving problem which is level 1 students always struggle with.

She would recommend students in her group to use this tool, if new example was delivered.

It is better to bigger size of the answer dialog box for inputting.

MSc Students Suggestions

3. The subject is an MSc student and used the Warm Up Demo and the BuildER examples to test the tool. She used manual control mode to play the worked examples.

She suggested that when the answer dialog box popped up, she would like go back to see the problem description text again, due to some texts were hidden. However, the tool did not allow her doing that and only choice was inputting the answer text. Although she thought the question very carefully and seriously, due to the limitation mentioned, it stopped her thinking and answering the question right. She had to press the "Cancel" button and the system assumed her did not know the answer for this question.

She asked why only allowing answering the question once, why not allowing the user to guess the answer first, and then comparing the guessing result with the module answer.

Correspondence function was found by told; due to she did not want to interact with the contents inside the example. She did not find glossary function until be told.

She also tried the tutorial and help function. She though the tutorial function was quite useful; but the help function was quite confusing, especially the address bar on the top of help window. She though the address bar was designed for searching help content, for example, entering a topic or some key words, the relative support documents would be shown in the help window.

The feedback view was too small. She thought it had the limitation of writing words, for example, it maybe only allow writing 20 to 50 characters. Meanwhile, the word "Your Feedback" caused confusing, due to she did not know whether designed to collect the suggestion of the tool, or the suggestion of the examples.

The contents in each step were fine. She did not need to remember too much in order to jump to the second step. Her eyes focused on changes in graphical panel first. Then she looked at highlighted text. Finally, she looked at graphical document panel.

She finally suggested that it would be better allowing user to make modifications on the graphical diagram and see the changes in the related panels automatically. After the modification, the editing result can be output to a standard database system.

4. The subject is an MSc student and used the Warm Up Demo and the BuildER example to test the tool. She used the auto play mode to play the examples. She did not understand the main GUI of the system very well, especially the explanation panel at the bottom of the GUI. She did not look at the explanation texts in the panel. She did think about the question carefully and seriously, but unfortunately did not notice the answer was recorded in the answer view.

Appendix 6

She suggested that maybe providing some guidance or demonstration of how the system works at the beginning. Meanwhile, she cannot distinguish the highlighted components in the graphical document. She realized that she should use the manual control of demonstrating. She especially liked the interactive idea of correspondence, although she did not find it by herself. She did not find glossary function either until be told.

Her eyes always focused on the yellow highlighted content first, if there was one. Then the line of sight would move to the explanation texts, finally watched the other changes in other panels. She explained that the yellow highlighting colour was so outstanding, so she always looked at first. However, she felt the highlighting colours confuse her during the demonstration, because it caused her to think the content using one highlighting colour was more important than another.

5. The subject is an MSc student and used the Warm Up Demo and the BuildER examples to test the tool. She used auto mode to play WarmUp Demo and manual control mode to play BuildER. She also reviewed the DB relation example, which called 1-1 relationship; due to she wanted to learn something she did not understand well.

She did not notice the change in the answer view at first, after inputting the answer. However, she found it after replying several questions. Then she run the same example again and suggested that she would like to see the teacher's model answer and her answer together in the same panel rather than looking at her highlighting answering first and then the teacher's model answer.

She actively found correspondence function at the end of Warm Up Demo example but bot the glossary function, and actively adjusted the panel and the view size in order to get enough space to view the BuildER example (like minimize the scenario view).

Her eyes focused on the changes in the graphical panel first, then the highlighting texts in problem description. Finally, she saw the explanation of this step. She preferred to use the auto play mode with longer waiting time for each step. If she wanted to continue to next step, clicking the "Next" button in order to go to next step.

She guessed the purpose of feedback view right and sent feedback messages successfully. Tutorial and help functions were found without problems.

9. The subject is an MSc student and used the Warm Up Demo and BuildER example to test the tool.

When running Warm Up Demo example, she used the auto play mode. When the answer dialog box popped up, she felt very uncomfortable. After several seconds, she found the question and focused on it, however, she did not think about the question very carefully. As she explained, this was only a warm up example, if it was in a real learning scenario, she definitely would take care about the questions. After clicking the "Cancel" button in the pop up dialog box, she did not see the changes happened in the answer view. However, when she clicked "Previous" button in order to go back several steps, she noticed the answer in the answer view was highlighted in green colour and realized that "I don't know" was the answer he make for that question.

She actively found the correspondence function through clicking the "Person" entity, but did not find the glossary function until told how to access it. She felt the glossary was easy to use and to follow, after pointing out how to access. She suggested it was better to give some tips or extra supporting information to guide the user using these functions. The glossary was the function she wanted, when using such kind of teaching tool to learn on her own.

When running BuildER example, she took this example seriously as a real learning scenario and used manual control mode for the rest exploration. She explained that the manual control mode was convenient to control the pace of learning, which was a better feature comparing with other teaching tools.

Her eyes focused on the changes in graphical document first, and then the highlighted text in problem description, finally the texts in the explanation view. However, if only the problem description highlighted texts and explanation texts, looked at the explanation first.

Sometimes she kept stay with the question step for a while, and then clicked "Cancel" button in the dialog pop up box. This behaviour was noticed by the observer and asked why? She explained that the answer was too long to type in and could see the model answer in the coming step, so just need to keep my answer in mind and compared it with the model answer.

Tutorial and help functions were found without problems.

Meanwhile, she found and pointed out the main GUI of this teaching tool was quite similar to Eclipse, which almost used every day, so she felt so familiar with it.

Appendix 6

She had no idea about the feedback view, but she guessed the purpose of designing the feedback view was right. However, she said even she sent feedback messages; she did not know where it was sent, who could receive it, and how to get the response.

She suggested that the answer and explanation views were needed to be even more outstanding.

She believed the explanations for this type of worked examples were so important.

10. The subject is an MSc student and used the Warm Up Demo and BuildER example to test the tool.

When running Warm Up Demo example, he tested both auto play mode and manual control mode and felt the manual control mode, using “Next” and “Previous” buttons were much more efficient than changing the speed bar in auto play mode. When the answer dialog popup box jumped out, he was looking for the question for a while. He always watched the changes happened inside the panels and then looked at the explanations. He also noticed the changes happened in the answer view, after replying question.

Correspondence and glossary function did not find until pointed out. He said he would not try to click the components in the graphical document, due to this size of example. However, if the example were bigger, he would try to interactive with the components in documents and could find the correspondence and glossary functions.

When running BuildER example, he used the auto play mode first, and then changed to manual control mode after replying the first question. He said it was quite hard to control the auto play mode, especially when there was a question. The speed of each step was too fast to think about the question. If changing longer time waiting time, some steps were not required. When he was unable to see the context of the question, he did not know what to do and asked for help. He also ask for how to get more space for the middle panel to see the example.

He thought about the question very carefully and used correspondence and glossary function for “Multi-Value” attribute.

His eyes focused sequence changed as well. At the beginning of the example, he looked at the graphical document change first, and then looked at the highlighted text, finally watched the explanation. While with the contents of the examples increasing, he looked at explanation first, and then looked at changes in graphical document, finally looked at the highlighted texts.

He found the tutorial function quickly, but did not the purpose of feedback view. However, he guessed the aim of feedback right and tried to write feedback messages and sent them away.

He found that using glossary function was very straightforward way to find what he wanted, and was very convenience for learning. Meanwhile, the dynamic procedure was the biggest advantage of this type of worked example comparing with examples in a textbook. The explanation explained the reason of changes in graphical document, which could help him to understand why doing that. He believed if there were examples for him to use in the real learning scenario, he definitely would use it to teach himself the contents, which not understand well in the lecture.

He guessed the purpose of feedback view right and sent messages successfully.

Level 1 students Suggestions

6. The subject is a Level 1 student and used the Warm Up Demo and CinemaExample example to test the tool.

He run the WarmUp Demo example using auto play mode first, but after several steps, he changed to manual control mode due to the auto playing speed was too fast. He tried correspondence function without any extra help, but tried glossary function with tips and suggestions.

He noticed the changes in the answer view after inputting the answer for the question, but he did not think the question very carefully.

He run the CinemaExample using the manual control mode and answered the question very seriously and carefully, because the content of the example was relevant to his study. He even tried to use the correspondence function in this example; unfortunately, the teacher did not setup the correspondence in this particular example.

His eyes always focused on the explanation first, and then looked at the new “Insert Content” with highlighting colour. He said the GUI was very similar to Visio Studio, so he felt so familiar with it and could finish the tasks very quickly.

He did not understand the feedback view and did not know how to use it. However, he guessed the reason of design this feedback function right. Then he tried to use this function to send a message. When asking why not try to use this tool in the lab, he replied that he was so confident with the programming course study and did not need any extra support, due to have VB and C language

Appendix 6

study and programming experience. However, if the lecturer could demo this tool inside classroom, he believed he would try it either in the lab or at home. He said the worked example shown in IWE was much better than the example in a textbook. It highlighted the linkage between problem and the solution. It saved time for self-identifying the relative contents in the answer. For example, in some programming textbook, the code is very long and it is hard and takes time to find the useful bit code when reading the book his own.

He also suggested that at the beginning of the explanation, it was better to mention that this solution was one possible solution and this question may have varied right solutions. However, this solution was the teacher's solution when he dealt with such kind of problem.

He found tutorial and help function s without problem.

He suggested that the scenario view could be minimized automatically when playing a worked example. This could bring more space for looking the contents.

12. The subject is a level 1 student and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he used auto play mode by default, however, only one step later, he changed his mind and began to use manual control mode to look at the example. He explained that step-by-step was good for learning and he preferred self-control the learning pace. He did not notice the question first, felt a little nervous when the answer dialog popped up. He did not notice his input answer was recorded in the answer view.

He actively found the correspondence function and believed it was a helpful feature, however, he could not find the glossary function, until mentioned right clicking the mouse on a piece of graphical document.

When running CinemaExamp, he tried to change the size of the panels in the main GUI. He did not think to adjust the size of scenario view and answer view, did not drag the bar between the panels in the middle of the GUI. He clicked the little arrow on the bars, which used to divide the middle area of the GUI.

He found he had to scroll to the top to see the problem description from the beginning, which was not very convenience. He clicked the content in the problem description document, and the text was highlighted in pink colour. He failed to unhighlighted his selection, until was told how to do that. Based on this point, he suggested that it may be better to allow the user to draw some lines on top of the problem description, which was like drawing some lines for highlighting. Later the user can compare his drawing with teacher's picking.

Sometimes he would like to go back to see the problem description again in order to write answer in the popped up dialog box, unfortunately, the system does not allow.

He compared his answer with the teacher's model answer, he found answering a question wrong and clicked "Previous" button to see the question again. Unfortunately, he clicked "Go to Begin" button by mistake, so he had to click "Next" button for 17 times to jump to his expected step. Hence, he suggested that it would be better to allow the user clicking once and jumping to the target step.

His eyes always watched explanation first. However, if there were highlighting texts, his eyes focusing sequence was changed with the content changing. For example, in step 28, he watched problem highlighting texts first, and then looked at the plan highlighting text, finally saw the explanation. In step 34, he watched the highlighted code fragments first, highlighted plan text second; the highlighted problem description texts third, if there were some.

He found the tutorial function slowly and found the help function quickly.

He guessed the purpose of feedback view wrong, after explaining, he sent feedback messages.

He said he really like the tool and would like to use the tool to see those examples designed for CS1Q either in the lab or download to his machine.

13. The subject is a level 1 student and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he used manual control mode. He actively found the correspondence function, but failed to find glossary function. He noticed that his replying for the question was recorded in the answer view. He never learn SQL before, just by using the warm up demo, he answer the testing question right.

Appendix 6

When running CinemaExample, manual control mode was selected. He clicked on the content of the problem description by mistake and some texts were highlighted in pink colour. He failed to unhighlight these texts, until told how to do it.

He thought the question carefully and seriously, and compared his answer with teacher's model answer as well. He suggested these two answers should be combined and shown together to make the comparison easier. He also wanted some extra explanations for step 40 which could explain how the "print msg" can be linked or mapped with the plan.

His eyes focused on explanation texts first, then the highlighting code (if there was), then the highlighting texts in plan, finally looked at the highlighting texts in problem description.

He found the tutorial function was hard to find, but the help function was easy to find. He found a bug for help window, which was missing the button for full size screen.

He guessed the feedback function right and sent messages, however, he did not know where the messages would go, who would response to it and how he could receive the response. He believed that the idea of sending feedback messages was good, but the implementation of this idea was poor. He never touched the "Play" button during the whole testing. He explained that he took the advice at the beginning of the example in explanation view, meanwhile, he preferred self-control his learning. Thirdly, the time for watching each step was not liner; it was quite hard to set a suitable time before watching the example. So using manual control for the first time viewing could be convenience. He said he may use auto play mode for his second, or third time in reviewing the example, which was like watching a movie.

14. The subject is a level 1 student and used the Warm Up Demo and the ERtoTables examples to test the tool.

When running Warm Up Demo example, he used auto play mode to watch the demonstration. He answered the question right, although he never learnt SQL before. He did not notice his answer was recorded in the answer view and mentioned the glossary function was easy to use, but hard to find unless be told. He found the correspondence between the contents in the two panels, after giving tip on the example was interactive worked example.

As he mentioned he had problem with the relationship from ER to tables in the class test. He did not performed well in the class test exam and would like to learn this bit, so using ERtoTables example instead of CinemaExample to test the tool.

He used auto play mode watch first, and then manual control mode to play the example each step again very carefully. After watching the example, he was actively interactive with the content, using correspondence and glossary functions. During the demonstration, his eyes focused on highlighting parts in ER diagram first, and then looked at the changes in tables, finally read the explanations. After watching the example, he mentioned he definitely learnt something, which he did not understand well before. If he saw this example before joining the lab, he would not ask his tutor how to deal with "Multi-Value" attribute in ER diagram.

He found the tutorial and help functions without any trouble and said they were easier to be found than the glossary.

He guessed the purpose of feedback function right, however, he used it in a wrong sequence. He clicked the "Send Feedback" button first without writing any message. He thought clicking that button was the starting point of sending feedback message.

When asking why not using it in the lab, he explained that he did not know there was a tool, which can be used. Meanwhile, he always figured out the problem by himself using Google. He also mentioned that he liked to use this type of interactive worked example and he would go back to the lab to see other examples.

16. The subject is a level 1 student and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he used auto play mode. He actively minimized the answer view and feedback view in order to get bigger area in the middle. He found the correspondence and glossary functions and noticed his answer was recorded.

When running CinemaExamp, he used manual control mode, due to requested using as a real learning scenario. He thought and answered the question carefully and seriously and compared his answer with the model answer as well.

Appendix 6

His eyes focused on explanation texts first, then the highlighting texts in the plan (if there was), finally looking at something other highlighting texts at most of the time. If highlighting code and plan texts together, he saw the code first.

He found the tutorial function quickly and used the feedback function correctly to send feedback messages.

17. The subject is a level 1 student and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he used auto play mode. She did not find the correspondence and glossary functions and did not notice her answer was recorded.

When running CinemaExamp, she used manual control mode, due to she wanted to use it to learn something. She thought and answered the question carefully and seriously and compared her answer with the model answer as well.

Her eyes always focused on highlighting part first, and then looked at the explanations. She thought the highlighting parts should be much more important. If there were more than two highlighting, she saw the plan first and then the code; or problem first and then plan (she already remembered the plan).

Tutorial was found, but took a longer time than expected. Help function was found quite quickly. She guessed the purpose of feedback view right, and test to send feedback message successfully.

She suggested that changing the explanation text position or colour in order to emphasize some important contents. This could help her focusing while watching the worked example to learn.

She liked the help content very much. She said the help was good enough to guide how to use the tool.

She also suggested sometimes the screen was too full of writing, maybe reducing some texts or using arrows to point.

18. The subject is a level 1 student and used the Warm Up Demo and the CinemaExample to test the tool.

When running Warm Up Demo example, he used auto play mode and changed the waiting time for each step frequently. He actively found the correspondence and glossary functions and noticed his answer was recorded. He read the explanation very carefully and opened the help content as suggested. He also tested the manual control mode, when doing some review. Using “Previous” and “Next” button to navigate the demonstration.

When running CinemaExamp, he used auto play mode with 5 seconds waiting time per step.

However, after he saw too many texts in the problem description in step 1, he changed to manual control mode. He thought and answered the question carefully and seriously and compared his answer with the model answer as well.

When coming back from step 8 to step 6, the green highlighting colour in answer view confused him. This colour made him feel he answered the question right, however, he saw the answer and knew he made a mistake.

He tried to highlight some useful information in problem description; the pink highlighting colour confused him. And he did not know how to unhighlight the selection, until told how to do it.

He clicked “End” button by mistake, and there was no cancel option for him to use. In order to recover to the original position, he had to click “Cancel” for all the questions left and then restarted the browsing.

He suggested it would be better to allow the user jump to a particular step, after these steps had been seen and allow the user to make some drawing on the top of problem description.

His eyes always focused on explanation first, and then the highlighted texts. If there were more than one highlighting, he saw the plan part before the problem description; or saw code first, due to the plan was in mind and familiar with it.

He found the tutorial function quickly and noticed a bug in the tutorial window, which was on the top of tutorial tab, text input was allowed.

He guessed the purpose of feedback view right, but not sure how to use it and how this feedback loop works. He also suggested it might be better to allow the user clicking a position in the panel, and then beginning to write feedback message based on this selected piece. Meanwhile, it was better to combine teacher’s model answer, user’s answer with the questions together. Otherwise, the user had to jump between the question and the answer.

Appendix 6

19. The subject is a level 1 student and used the Warm Up Demo and the CinemaExample to test the tool. When running Warm Up Demo example, he used manual control mode. He did not notice his answer was recorded and mentioned the correspondence and glossary were hard to find. However, after been told, it was easy to use and very convenience for learning.

He believed the glossary function should be a part of this teaching tool. He thought about the question very carefully and looked at the explanation very carefully as well. He found a spelling mistake in step 9 which happened to the word “example”.

When running CinemaExamp, he used manual control mode. He thought and answered the very carefully and suggested that step 14 was a confusing question and in step 16 he would like to go back to see the problem description again, but the system did not allow. Hence, it was better to allow the user to see those texts again, when answering a question. Meanwhile, sometimes one step provided so much information, made him feel hard to remember all.

His eyes focused on highlighting texts first, if there was one. Then looked at explanations. If there were more than one highlighting, he preferred looking at the bright colour first.

He found the tutorial and help functions quickly and looked the contents as well.

He guessed the purpose of feedback view wrong, but tested to use the feedback function to send message without problem.

Appendix 7: Teacher's Evaluation Report

Context

Both lecturers were mid-flow through their courses and so wanted to create worked examples that would fit directly into their teaching at the current point reached. Given their busy timetables, there was little time to spend on practice examples and instead, they both launched into relatively complex examples to suit their course, without having fully explored manuals, tutorials and so on. One teacher commented "I rarely read manuals for software nowadays, I can usually work it out by playing with the tool". Such an attitude depends on the new software using similar techniques and approaches to existing software. Furthermore, given their relative inexperience with IWE, the lecturers came with ideas and expectations of what they wanted to achieve that were not ideally suited to what IWE could offer.

This document is a summary of the experiences of the teachers, written by them.

Workflow

There was a level of acclimatisation to the workflow model in IWE, given the lecturers' previous experience with WYSIWYG development tools like word processors and drawing or presentation packages. In those tools, it is customary to generate the content within the tool and, at the same time or subsequently, style and structure it as necessary.

By comparison, IWE was not originally envisaged as a content generation tool. The ideas for IWE formed around the need to demonstrate more clearly to students the manner in which one, for example, software engineering, document was transformed into another. The point is that the source and destination documents were already in existence, and the challenge was to show how the transformation worked. With that perspective, IWE is a tool for structuring existing documents so that they can be presented as interactive worked examples with a sequence of steps and integrated explanations.

The two lecturers found that the need to create fragment and then document types at the very start of the process confusing. In the light of the workflow model, however, this can be explained: the lecturers expected to use the tool for content creation as well as styling and structuring, whereas the tool starts from the point of view of styling and structuring, assuming content creation has already been completed outside the tool. If styling is one of the first things to be done, as text and graphical fragments are defined, then the fragment/document types must have been created beforehand.

Document Structure

The idea of a graphical document being made up of fragments is easy to appreciate, since drawing packages have a similar model - for example, in an ER diagram, an Entity is a fragment and its type defines its appearance as, for example, a box with a label enclosed within it.

Considering a text document as a sequence of fragments is more unusual, however, and was a source of difficulty for the lecturers initially. In the examples used, the fragments of text are often smaller than a single paragraph or sentence. The user must determine this fine-grained breakdown of the text

Appendix 7

into these small fragments, in order that each can have the appropriate style, or fragment type, applied. Fine-grained styling of text is not unusual in text editors, but it is usually built-in rather than user defined - consider the typical built-in bold, italic, underline text styling options. In IWE, these need to be defined as fragment types, much as Word styles are created. From the point of view of transfer from other packages, the most commonly used Word styles are so-called 'paragraph' styles that apply to a whole paragraph, not to a word or small portion of a sentence. Such styles do exist in Word - the *character* style type - but they are used only infrequently.

Particular unexpected characteristics noted include:

1. **Forcing new lines.** A text document is simply a list of fragments. A method is required to denote in a fragment that a new line is required, and the method chosen was unfamiliar to the teachers. To force a new line to be displayed after a fragment as the worked example is viewed, a semi-colon must be added at the end of the fragment. Provided this is the last character of a fragment, a new-line will be forced. The semi-colon is only treated as a new line character if it is the last character in the line, so semi-colons can be embedded in a line of text and if a semi-colon is required at the end of a line (e.g. when creating a line of code in a programming language such as Java) then ';;' must be used at the end of the fragment, the first semi-colon will appear as part of the text and the second one will be treated as a new-line character. When importing text a common error is to create one or more spaces after the semi-colon - which means that it won't be treated as a new-line character.
2. **Lack of layering.** The layering of components in a graphical document was queried. While even primitive drawing packages support a layering model (all objects can be thought of as being in a stack), IWE does not support this. With a layering model, simple animation can be supported by making visible one by one a series of overlapping objects in the stack. Each one obscures the previous object, and one has a kind of "stop-frame" animation as each new object is viewed. There is a level of cognitive dissonance emerging here between the worked example presentation model offered by IWE and the animation model offered by traditional software, e.g. Powerpoint.

The present implementation of IWE does not permit new fragments to be inserted into a document as a process is defined. It is possible to edit the content of a fragment but in general it is not possible to split an existing fragment into two or more parts. Therefore, the complete structure of each document has to be defined before starting to define the process that shows the transformation between the documents and adds the explanations.

For example, when developing the pseudocode for a programming example, the following fragment was defined:

```
while <there are more lines> do
```

However, when defining the process, it was realised that ideally the condition needed to be highlighted separately:

```
while <there are more lines> do
```

This is not possible because highlighting can only be applied to a complete fragment, and IWE does not currently allow the single fragment shown to be split into the three fragments necessary - "while", "<...>", and "do".

Format of, and Control While Viewing, Worked Examples

In general, the two teachers wanted greater control over the way that the worked examples were presented in IWE.

The teachers wanted additional options for formatting the text displayed in various areas of the IWE interface. For example, there is no ability in textual documents to represent items in a neatly formatted list, or to use tab characters, particularly useful in, for example, presenting code fragments. Similarly, the explanation text would benefit from formatting and also from the ability to add hyperlinks to other explanatory materials, should the student need further information.

When defining the presentation of a worked example, the user is given a choice of possible layouts for the panels that display the documents. However, each document pane is given a default size within the display area so that space is shared equally between panes. The teacher cannot change this default layout, which would be useful in some cases. For example, if there are two panes, one containing text and the other a diagram, then it may be desirable to give a larger proportion of the display area to the diagram. The CinemaExample used a layout with three adjacent vertical panes but the first part of the process focused on only two panes (left and middle) and the second part of the process on two panes (middle and right). It is possible for students to adjust the size of panes and so hints were given to students to adjust the relative sizes of panes via the explanation panel. However, an improvement would be to allow the teacher to include instructions in a process to automatically adjust the size of the panels.

Similarly, teachers found large bodies of text awkward to handle. Whilst an explanation might be referring to a particular section of the text, the teacher currently has no control over whether that text will be visible at the time the explanation is read, even if the text fragment itself has been explicitly highlighted. This is likely to be very confusing to a student who is trying to follow the explanations and who must link them up to the documents involved. Options for change here are to ensure that the most recently highlighted text appears in the centre of the panel in which it is displayed, or else to enable the teachers to add an instruction to the process that sets what text is to be visible in the panel.

It would also be useful to allow multiple highlighting colours. In the example below of a fragment of a problem description showing how some sample input maps to the corresponding output, five lines of input and the corresponding output are highlighted, to go with an explanation encouraging the student to focus on these. The next step then gets the student to focus on one particular input/output pair - the fourth one down, as a particular example of where a booking request could not be satisfied. The example shows how two highlighting colours are of value - by retaining overall focus on the sample input/output section while concentrating on a particular pair.

For example, if the following requests were typed in:

```
3 c 25
4 f 28
3 f 5
4 c 3
0
```

then your program should output the following messages:

```
Request accepted
```

Appendix 7

```
Request accepted  
Request accepted, theatre 3 now full  
Request denied, only 2 seat(s) available in theatre 4
```

```
Total takings: 215 pounds
```

The question construct forces an unnatural style of questioning because the context for the question has to be set up before the question is asked and then is invisible once the question is posed and the system is awaiting the student's answer. Asking the question takes precedence over any other actions in a step, such as highlighting of fragments related to the question. Also the pop-up box for the answer obscures part of the context of the question. The pop-up box can be moved around to expose the contents of other panels but panels cannot be scrolled while the pop-up window is awaiting input.

Suggestions for the IDE

The definition of processes can be tedious as there is no mechanism for selecting a group of fragments and highlighting them or revealing them in one operation. Similarly, to remove the highlighting each fragment has to be unhighlighted in a separate operation. A grouping mechanism is required, similar to those found in any graphical editor.

As an example, the same section as above from the CinemaExample is used. The highlighting of the sample input and output occurs in a single step. Each of these lines is a separate fragment, forced by the newline mechanism, and so to highlight all the lines requires ten individual highlight operations, one for each fragment. The corresponding unhighlighting requires another ten operations.

For example, if the following requests were typed in:

```
3 c 25  
4 f 28  
3 f 5  
4 c 3  
0
```

then your program should output the following messages:

```
Request accepted  
Request accepted  
Request accepted, theatre 3 now full  
Request denied, only 2 seat(s) available in theatre 4
```

```
Total takings: 215 pounds
```

If a grouping mechanism were in place, two groups could be created, for example, for the input and the output respectively. This would retain flexibility for referring to each separately while reducing the number of operations needed significantly.

The teachers noticed an annoying glitch with the explanation entry panel - they had to explicitly press a "Update" button before moving to the next explanation, or else their text was lost with no warning from the system. This happened frequently.

Bringing in text created using other tools was sometimes noted as problematic, as the character sets did not always match well. Spurious characters appeared in both fragments and explanations when text was imported or copied/pasted into IWE. While some of these inconsistencies have been resolved, a few are outstanding and require careful checking.

Updating documents and processes

A major feature of IWE is the ability to change worked examples in the light of feedback from students. While the teachers did not in the end receive significant feedback from students via the tool, there was extensive checking/review of the worked examples before they went out to students. This required repeated updates to be made to the worked examples.

There are limitations to updating worked examples, as described in the *Document structure* section above. Apart from these however, the teachers were able to make changes to documents, process steps and explanations quite easily and promote those out to the students. This could have been on the basis of student feedback, just as it was on the basis of the teachers' own critical feedback.

Although the formal feedback mechanism didn't receive large use by the students, many students did answer the questions. Considering the questions answered by students on worked example Unit8-task2, the lecturer is able to draw much feedback. Question 1 for example is answered incorrectly by the majority of students, which could be a cause for concern, but in fact appears to simply be effective at catching a misunderstanding, and the explanation should help. Question 2 is answered surprisingly badly, pointing possibly to a use of terminology that the students weren't familiar with. The question text could be altered to "what function header is required - that is, what would the first line be?" Question 3 has a range of different answers, but checking by eye shows that they are all reasonable answers - giving a sensible name for the function. This is a case which exemplifies why automatic answer checking or marking by the system would be challenging. Question 4 is also poorly answered, although the wording ("What, if anything, should now come after the name sumList?") does not seem overly confusing. This may be a case for additional work in a lecture, or a message out to students. In any case, it is clear that the teacher can draw much from these answers with which to improve either the worked example or his/her presentation in general.

Representing evolving documents

Some of the programming examples presented a problem and a *plan* to solve the problem - two documents. The teacher wanted to use IWE to model the development of that plan. The typical model in IWE is to imagine how one document transforms to another - in this case how the problem leads to a particular plan for solving the problem. The teacher wanted to use the stepwise refinement technique, where a high-level version of the plan is first developed, and then this is steadily "refined" to give steadily more detailed versions of the plan. Although this was not realised initially by the teacher or the researcher, the teacher was in effect creating multiple documents, one for each refinement of the plan - but using IWE in such a way as to contain all these versions in a single document.

A very simple example of this is given below:

The problem is: Write a program to read in numbers until -1 is entered.

The total of the numbers read in, not including the -1, should be output at the end.

A top level plan for this problem could be:

```
<initialise variables as necessary>
while <more lines to process>:
  <process next line>
```

Appendix 7

<write out total>

This is a complete plan that solves the problem, albeit at a high level of abstraction. Having shown this solution, it would typically be refined, by for example replacing each of the elided sections, e.g. <...>, with a more detailed section.

```
set total to zero
set inp to the first number read in
while <more lines to process>:
  <process next line>
<write out total>
```

So far, only the first elision has been expanded. To model just the first two lines and their subsequent expansion into three lines, the following fragments are required:

1. <initialise variables as necessary>;
2. set total to zero;
3. set inp to the first number read in;
4. while <more lines to process>;

The steps required to first show the elided line and the while loop header, and then to expand the elided line are as follows:

```
S1 Insert 1          # the elided line
S2 Insert 4          # the while loop header
S3 Delete 1, Insert 2 # delete the elided line, replace with the first line of
the expansion
S4 Insert 3          # the second line of the expansion
```

The solution works, but is certainly not the initially expected use of IWE. The textual document created for the evolving plan is a merging of a series of documents and not a document in its own right. Furthermore, it is intimately linked to the process that displays it, reducing the likelihood that it would be reused in other processes - an initial aspiration of IWE. (*See the CinemaExample which uses this technique at the end of this report.*)

Techniques for Delivering Certain Sequences in Worked Examples

The teachers at one point determined that they wanted textual fragments to appear out of order. This presents no difficulty when the fragment is a whole line, but is trickier when the fragment is part of a line.

The teachers' particular example involves an alternative way of representing the evolving plan for a program, this time using a true single document. The difference is that each line of the top-level plan is refined in a separate section below the top-level plan. The top-level plan is largely unchanged - it simply acquires a "refinement number" next to the line that is to be refined below. The refinements are headed with their refinement number.

To model this kind of evolving document, the first few steps make visible the lines of the top-level plan. No refinement numbers appear at this point - the top-level plan is after all complete as it is. However, the purpose of the worked example is to then show the refinements. This requires the addition of a refinement number next to the line to be refined - and it is this that necessitates the appearance of fragments out of order

Here is the full text of an example document of this kind:

```
Read in a number
Write out the times table for that number -- 1
```

Appendix 7

Print a completion message

Refinement 1

<the refinement lines>

The presentation sequence required has five steps, S1 to S5, as follows:

S1

Read in a number

S2

Read in a number

Write out the times table for that number

S3

Read in a number

Write out the times table for that number

Print a completion message

S4

Read in a number

Write out the times table for that number -- 1

Print a completion message

S5

Read in a number

Write out the times table for that number -- 1

Print a completion message

Refinement 1

<the refinement lines>

Some extra little textual fragments are needed to make this work properly. The fragments then become:

1. Read in a number
2. Write out the times table for that number
3. ; #Ensures lines 1 and 2 appear separately
4. -- 1;
5. Print a completion message;
6. ;
7. *Refinement 1*
8. <this fragment and following are the refinement lines>

The steps of the process required are now as follows:

- | | | |
|----|------------------------------|---|
| S1 | Insert 1 | # Displays line 1 |
| S2 | Insert 2, Insert 3 | # Displays line 2 adding in a new line |
| S3 | Insert 5 | # Displays line 5, skipping line 4 at this stage |
| S4 | Delete 3, Insert 4 | # Must delete new line, so refinement number is shown |
| S5 | Insert 6, Insert 7, Insert 8 | # Display the refinement itself |

IWE and PowerPoint

The teachers mused on how IWE compared with PowerPoint, the tool most often used to create worked examples. Typically, a diagram is built up using a series

Appendix 7

of slides, each the same as the previous one apart from one or more additional components; or else the animation feature is used on a single slide where a single complete diagram is revealed piece by piece.

This idea of revealing components is of course similar to the use of graphical documents in IWE. However, there is no easy parallel for IWE's documents consisting of fine-grained fragments. Certainly, PowerPoint can reveal a text item line by line, but it not possible to reveal or highlight a partial line fragments as is possible and clearly useful in IWE worked examples. To do so in PowerPoint would require all text fragments to be individually created and aligned and would become managerially extremely complex. Furthermore, the addition of large numbers of explanatory textual items, where they all on one slide, would also become unmanageable.

In short, while PowerPoint is used effectively for small-scale worked examples, the teachers could not imagine trying to use PowerPoint to create worked examples of the complexity they achieved with IWE.

Conclusions

This evaluation may seem critical, containing large numbers of suggestions for improvement. It is crucial to report however that the teachers were able to create a number of different styles of worked examples - Graphical to Text, Text to Graphical, Text to Text, and Text to Text to Text. Furthermore, these examples were updated in the light of critical review by the teachers, and were sent out to students for their successful use.

An overarching report from the teachers is that the *creation* of a worked example, involving the development of the documents and an appropriate fragmentation and set of explanations, requires much more intellectual effort than the *structuring*, whether on paper, PowerPoint, or using IWE. The latter process is essentially mechanistic once the underlying worked example content is clear. Furthermore, while the initial document creation and broad idea of the fragments, steps and explanations may be created prior to using IWE, the realisation of the worked example in IWE then prompts numerous refinement cycles to improve the worked example educationally. The whole process is lengthy, but as already stated, this is due to the creative effort required, not the mechanics of the tool itself.

CinemaExample

The CinemaExample required the Plan for the program to be presented as a series of refinements, conceptually seven evolving documents each becoming more detailed. IWE does not support the presentation of such a series of documents, as each would need to appear in a separate window, and with the Problem Description and Program, a nine pane display would be unmanageable. The Plan could have been presented in a single scrolling pane with lots of repetition of the text. However, a solution was found that involved creating one document and selectively displaying and removing parts of the document. This single document was effectively a merger of all the iterations of the Plan. How this works is illustrated below.

Overall Plan:

<Set up data storage> ---1

<Read in and process the lines of input, updating storage and writing out messages as appropriate> ---2

Appendix 7

<Write out the total money taken>

Refinement 1:

<Create a list of integers with five elements, all initially set to zero>

<Create an integer variable for the money taken, initially set to zero>

<Read in and process the lines of input, updating storage and writing out messages as appropriate> ---2

<Write out the total money taken>

Refinement 2:

<Create a list of integers with five elements, all initially set to zero>

<Create a tally for the money taken, initially set to zero>

<Read in the first line of input>

WHILE <next line is not the character zero>

 <Process the line> --- 3

 <Read in the next line>

<Write out the total money taken>

Refinement 3:

<Create a list of integers with five elements, all initially set to zero>

<Create a tally for the money taken, initially set to zero>

<Read in the first line of input>

WHILE <next line is not the character zero>

 <Unpack components of the booking>

 <Determine and carry out appropriate action> --- 4

 <Read in the next line>

<Write out the total money taken>

Refinement 4:

<Create a list of integers with five elements, all initially set to zero>

<Create a tally for the money taken, initially set to zero>

<Read in the first line of input>

WHILE <next line is not the character zero>

 <Unpack components of the booking>

 IF <not enough space for this booking>

 <Write out message to say that this booking can't be satisfied>

 ELSE

 <The booking can be satisfied - act accordingly> --- 5

 <Read in the next line>

<Write out the total money taken>

Refinement 5:

<Create a list of integers with five elements, all initially set to zero>

Appendix 7

<Create a tally for the money taken, initially set to zero>

<Read in the first line of input>

WHILE <next line is not the character zero>

 <Unpack components of the booking>

 IF <not enough space for this booking>

 <Write out message to say that this booking can't be satisfied>

 ELSE

 <Do the necessary updates and output for a successful booking> --- 6

 IF <the requested cinema is now full>

 <Write out message to say the theatre is now full>

 <Read in the next line>

<Write out the total money taken>

Final Plan (Refinement 6):

<Create a list of integers with five elements, all initially set to zero>

<Create a tally for the money taken, initially set to zero>

<Read in the first line of input>

WHILE <next line is not the character zero>

 <Unpack the components of the booking>

 IF <not enough space for this booking>

 <Write out message to say that this booking can't be satisfied>

 ELSE

 <Update the number of seats used in the chosen cinema>

 <Update the money taken to include this booking>

 <Write out message fragment to say that the booking is satisfied>

 IF <the requested cinema is now full>

 <Write out message to say the theatre is now full>

 <Read in the next line>

<Write out the total money taken>

All these refinements need to be merged into a single document in which the lines to be refined are interleaved with the refinements in the correct order for presentation. The Plan document for the CinemaExample is shown below.

Plan Document

1. <Set up data storage> --- 1
2. <Create a list of integers with five elements, all initially set to zero>
3. <Create a tally for the money taken, initially set to zero>
- 4.
5. <Read in and process the lines of input, updating storage and writing out messages as appropriate> --- 2
6. <Read in the first line of input>
7. WHILE <next line is not the character zero>
8. <Process the line> --- 3
9. <Unpack the components of the booking>
10. <Determine and carry out appropriate action> --- 4
11. IF <not enough space for this booking>

Appendix 7

12. <Write out message to say that this booking can't be satisfied>
13. ELSE
14. <The booking can be satisfied - act accordingly> --- 5
15. <Do the necessary updates and output for a successful booking> --- 6
16. <Update the number of seats used in the chosen cinema>
17. <Update the money taken to include this booking>
18. <Write out message fragment to say that the booking is satisfied>
19. IF <the requested cinema is now full>
20. <Write out message to say the theatre is now full>
21. <Read in the next line>
- 22.
23. <Write out the total money taken>

To create the *Overall Plan* requires the following sequence of operations for the Plan display pane:

Insert line 1
Insert line 5
Insert line 23

To create *Refinement 1*:

Delete line 1
Insert line 2
Insert line 3
Insert line 4
lines 5 and 23 remain displayed

To create *Refinement 2*:

Delete line 5
Insert line 6
Insert line 7
Insert line 8
Insert line 21
Insert line 22
lines 2, 3, 4 and 23 remain displayed

To create *Refinement 3*:

Delete line 8
Insert line 9
Insert line 10
lines 2, 3, 4, 6, 7, 21, 22 and 23 remain displayed

And so on ...

This approach is possible because IWE supports deletion of lines from a document display pane, not just insertion. The Back button on the tool bar allows a student user to see earlier versions of the Plan, if required.

Appendix 8 Lab Exercise Sheet for Week 7

Computing Science ICT Lab 7

You can start these before the lab – indeed, it can be helpful to do so, as then you can ask any questions that arise as soon as you come into the lab.

1. If you missed the Friday lecture, or if you are unclear how the "clock face" program was constructed, then first use IWE – the *worked examples* tool – to review how its development. To do this, either use the *IWE* shortcut on the desktop in the lab, or if you're at home, download the tool from the following address:

<http://www.dcs.gla.ac.uk/~yulun/Tool/IWE.rar>

It's a Java program, so you will need a Java run-time installed on your machine – try double-clicking on the file to find out. In the left-hand list, double-click on *ClockExample*, and then on *DrawingClockProgram* to start viewing the example.

2. Now solve the following problem:

Write a program to draw a bicycle wheel – that is, a wheel rim and, say, 8 spokes running from the centre of the wheel out to the rim.

This is related to the clock-face program. The challenge here is to see the relation. If it's not obvious, stick with it. How could the tools you've seen to draw a load of numbers in a circular formation be helpful in drawing the bicycle wheel?

3. We're now going to work with another problem:

Draw 10 parallel horizontal lines, each 100 long, starting at (10,10) and spaced 10 apart, going downwards.

For some of you, it will be obvious how to do this. For others, it may not be. This is normal! If you're in the latter category, you could play around with the problem by getting pen and paper out – draw up some lines... how are they related?

If it's not making sense, use the IWE tool again – this time, view *HorizontalLinesProgram*. Use it to get the smallest hint – as soon as you can see how to go forward, do so.

Review the IWE example at the end to see how your solution compares to this one.

4. Develop a solution to the following, related problem:

Draw a grid of cells (like an empty table) with 5 cells across and 4 cells down. Each cell should be 30 wide and 20 tall.

See if you can solve this with no help first. It's obviously related to the previous problem, but how? Only read on if you want some clues...

Hint: You could draw lines or draw boxes. The lines version is related to (3) – start by changing your program to draw horizontal lines so that it draws vertical lines. Once you have that working, decide how you should combine the two programs to draw a grid.

5. *Write a program to draw a random segmented line. More precisely, the line should start at (100, 100) and consist of 10 straight segments, connected so that the end of the first line joins up with the start of the second, and so on, to form one long line. The length and orientation of each segment should be chosen at random, between the limits 0-50 and 0-359 respectively.*

Appendix 8

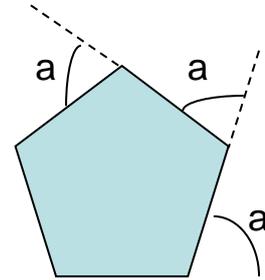
To generate random numbers, you should import the **random** module. Inside, there is a function `randint(a, b)` which returns a random integer N such that $a \leq N \leq b$.

If you need some hints, then check out the IWE worked example *RandomLineProgram*. As before, only view it as far as you need until you have an idea how to continue.

6. Write a program to draw a regular 5-sided shape – a pentagon.

"Regular" means that the sides are all the same length, and the angle between any two adjacent sides is the same.

There are different ways of solving this problem. One is related to the previous problem.



The diagram on the right may help as you consider how to solve this. The angle a , between the extension of one side and the next side, is the same all the way round.

7. Now adjust your program so that it can draw *regular* shapes of any *size*. You should turn your drawing code into a Python function that takes four parameters – one each for the x,y position of the start of the drawing, one for the number of sides you want the shape to have and the other for the length of each side.

Hint: consider which values used in the pentagon example are a result of the fact that it's a pentagon we're drawing. How can the equivalent value be calculated for a shape with more or fewer sides?

If you're unsure here, check out the IWE worked example *PentagonToShapeProgram*.

8. Go back to your program to draw a grid, and turn that into a function as well. What are the parameters that you need to specify an arbitrary grid to be drawn? Are all of the parameters essential? Or, could you make some of them optional, in the way that, for example, the *fill* parameter in *create_line* is optional?
9. Before calling your tutor over to look at your work, consider the following checklist,
- Style – are the variable names you've used helpful in understanding the program? Is the code laid out neatly – is it easy to read? Have you put appropriate (not excessive) commenting in?
 - Understanding – do you fully understand how all parts of your program contribute to solving the problem?
10. **Harder/Optional:** Write a program that draws the movement of a position on the rim of an imaginary wheel as it 'rolls' across the screen for 5 revolutions. This is a bit like *spirograph*, if you had that toy as a child.

If you're having trouble imagining what this should look like, take a large round coin (£2 or 2p?) and roll it along a straight edge. Hold a pen/pencil against a point on the edge of the coin, and draw a line as the coin rolls.

Appendix 8

11. Further challenges, for your interest:

- a. Draw a spiral (or rather, an approximation using many straight lines!)
- b. Draw a 5-pointed star, and then extend this so that it will draw an n-pointed star, where n is read in from the user

Use the *wait* function in the *Canvas* library to put a delay between the drawing of each line in your diagram – your drawing will then be animated.

Appendix 9 Student Lab Exam Question

Computing Science 1 Laboratory Examination 2013-14

2nd - 4th December 2013 Time allowed: 1 hour 50 minutes + advance preparation.

Weight: the exam contributes 10% to the overall assessment of your course.

Threshold: Students must attempt at least 75% of all assessments on the course to gain credits for the course. This exam contributes 10%.

Lab exam problem

Context

The web crawlers used by systems such as Google make use of an *indexer* that picks out certain words from the web pages that the crawler discovers and puts these into a huge index used later by the search engine. From all the words on the page, the indexer chooses which words to put into the index using a range of techniques. This exercise involves two particular techniques:

24. **Stop word list:** common or so-called *stop* words like a, the, is, and, so, then, etc., are held in a stop word list. All occurrences of these common words found on the page are ignored from the point of view of building the index.

25. **Stemming:** many words have the same *stem* but different *endings*, e.g. cook, cooks, cooked, cooking. To keep the index small, words are checked to see if they have one of the common endings like -s, -es, -ing, etc. and if so the word is *stemmed*, i.e. the common ending is removed.

After removing stop words and then stemming those that are left, the remaining words are put into the index, with references to the web page they appeared on.

Problem

You are to write an indexing program that will record and print out on which *lines* particular words appear in a piece of text supplied as input by the user. Hence, the index you generate will look like a book index, but each index entry will have a word followed by the *line* numbers on which the word appears, rather than the *page* numbers. Specifically, your program should:

26. read in lines of text one at a time, keeping track of the line numbers (counting up from 1), stopping when a line is read that contains *only* a single full-stop;

27. remove punctuation (as specified below) and change all text to lowercase;

28. remove stop words (the stop word list is specified below);

29. stem the words (the common endings to look out for are specified below);

30. add the remaining words to the index – a word should appear only once in the index even though it may appear many times in the text, and the line numbers on which it appears (removing duplicates) should be recorded with the word;

31. print the index, **using exactly the format below**, once all lines have been entered.

Particulars

Appendix 9

3. Some of this is similar, but not identical. Bear in mind that strings are like lists of characters, and that sentences are lists of words. You can manipulate them in much the same way... removing punctuation, removing stopwords...
4. If you don't know how to use the *split* function, found in the string module, find out.
5. You could use a dictionary or a list to hold the final index.

Appendix 10 Related Questions in Student Written Exam



University of Glasgow | School of Computing Science

18th December 2013
13.00pm – 15.00pm
(Duration: 2 hours)

CS-1CT Introduction to Computational Thinking

25. The following Python code is supposed to add up ten numbers entered by the user. All numbers should be within the range 0 to 100 inclusive. If any number entered is outside this range, the number should be ignored and the user asked to type in another number. It contains three errors. Give a clear explanation of each error and say how to correct it.

```
count = 0 #1
while count < 10: #2
    n = input( "Please type in a number: " ) #3
    if 0 <= n or n <= 100: #4
        total = total + n #5
    else: #6
        print( "Out of range, please enter again" ) #7
    count = count + 1 #8
print total #9
```

[6]

26. This question involves a dictionary storing information about a shop's stock. The dictionary is indexed by product names, represented by strings. Each entry is a dictionary storing information about the product, again indexed by strings, including a floating point item with the index "price" and an integer item with the index "quantity". The structure of the dictionary is shown schematically as follows:

```
["baked beans": [{"price":0.39, "quantity":40, ...},
"sliced bread": [{"price":0.64, "quantity":25, ...}, ... ]
```

The following function is supposed to calculate the total value of the shop's stock. It contains three errors. Give a clear explanation of each error and say how to correct it.

```
def totalValue(stock): #1
    for item in stock: #2
        value = value + stock[item][price] #3
    return value #4
```

[5]

27. Make sure you read the whole of this question, on this page and the next, before starting to write any code.

You are to write a complete Python program which allows the user to type a string at the keyboard and then displays in four columns how many times each letter of the alphabet occurs in the string. Assume that all letters entered will be in lower-case, and ignore any other characters.

For example, if the input is

today is the day of the cs1ct degree exam

then the output should be in four columns as shown below. Note that perfect alignment of the columns is not essential, but using this input as an example, there should be six lines of four items and one line of two items.

```
a: 3  b: 0  c: 2  d: 3
e: 6  f: 1  g: 1  h: 2
i: 1  j: 0  k: 0  l: 0
```

Appendix 10

m: 1 n: 0 o: 2 p: 0
q: 0 r: 1 s: 2 t: 4
u: 0 v: 0 w: 0 x: 1
y: 2 z: 0

When writing your solutions to the questions below, be sure to use good style –variable names, layout, etc. – and please make it as legible as you can.

- a. Define a function that takes a character and a string as parameters and returns the number of times that the character appears in the string. [5]
- b. Using the function defined in (a), write the complete program. You may prompt for the input in any way you choose, but you should ensure that the output is in the format described above. Note, you don't need to re-write the function you created in (a). [9]

Appendix 11 The Implementation Structure of IWE

In order to achieve and implement the data model of IWE, a UML static and hierarchical structure for guiding the development was designed. Figure 5.1 shows the UML class diagram of IWE with detailed implementation information. It maps the three-layer data model of IWE, given in the Requirements Chapter, into a class structure.

Document Type is an abstract class, which maps to the Document Types layer in the data model. It is inherited by *Graphical Document Type* class, which holds the different graph structures and *Textual Document Type* class, which holds the different textual styles. *Graphical Fragment Type* class is designed to describe a single graph structure, which could be a node (shape) or an arc (line). *Textual Fragment Type* class is designed to describe a single textual style.

Graphical document type contains a collection of graphical fragment types and textual document type contains a collection of textual fragment types. Due to some common features shared between graphical fragment type and textual fragment type, a *Fragment Type* interface is provided.

Document is another abstract class, which maps to the Document layer in the data model. It is inherited by *Graphical Document* class, which is an instance of one type document and consists of a collection of graphical *fragments*. *Textual Document class* is an instance of another type document and consists of a collection of textual *fragments*. These fragments depend on related fragment types, which mean one fragment type can generate several fragment instances. Document is made up of fragments, and fragments are instances of fragment types, document type is made up of fragment types, so document depends on document type and can also be understood as an instance of a document type. Meanwhile, due to some common features shared between graphical fragment and textual fragment, a *Fragment* interface describing the most general methods of a fragment, is included.

The *Application* class maps the example set layer in the data model. An application can be understood as a container, which holds a set of documents as resource to create a collection of processes. Creating an application means combining several documents. Because an application can use several documents in order to be presented as multiple views, it requires a container to hold the documents. The *Application Panel* class is designed to achieve this purpose. Application not only manages allocating documents into application panels in a predefined multiple-panel structure, but also controls setting up the relations between different fragments in different documents, the *Correspondence* class is provided to manage this relationship.

A process can be understood as a worked example. Figure A11.1 also indicates the relationship between an application and a process. An application contains a set of processes, where each process defines a different worked example based on the same set of documents involved in an application. Hence, a *Process* class is designed to represent the requested fragments of the documents, which are involved in an application. It contains an array of steps, and each step contains an array of changes and an explanation for these changes. So the *Step* class, *Change* class and *Explanation* class are provided.

The purpose of process is to show the required fragments of documents step by step with explanations, so a process class is designed as a part of an application. In order to show the required fragments of documents step by step, a step class is required as a part of the process to manage displaying the contents of documents and explanations. The step class can be understood as a container of changes with the unique explanation of these changes.

A change class is designed as a part of a step, which holds the references to the fragments in a document with the operation to be carried out on them. Hence, a change can be understood as a fragment in a document plus the operation on this fragment, which could be insert, delete, highlight, unhighlight, show whole document and ask a question. Figure 5.1 demonstrates the relationship between a change and a step, a change and a document, a change and a fragment as well.

In order to explain the changes that happen in a step, an *Explanation* class is designed as a part of a step as well. The explanation class manages the texts, which are associated with a particular step. Because a step may contain several changes that happened in the different application panels, the explanation can emphasize the relationship between these changes in a step.

This detailed design achieved all the requirements in the data model of IWE and guided the final development and coding work.

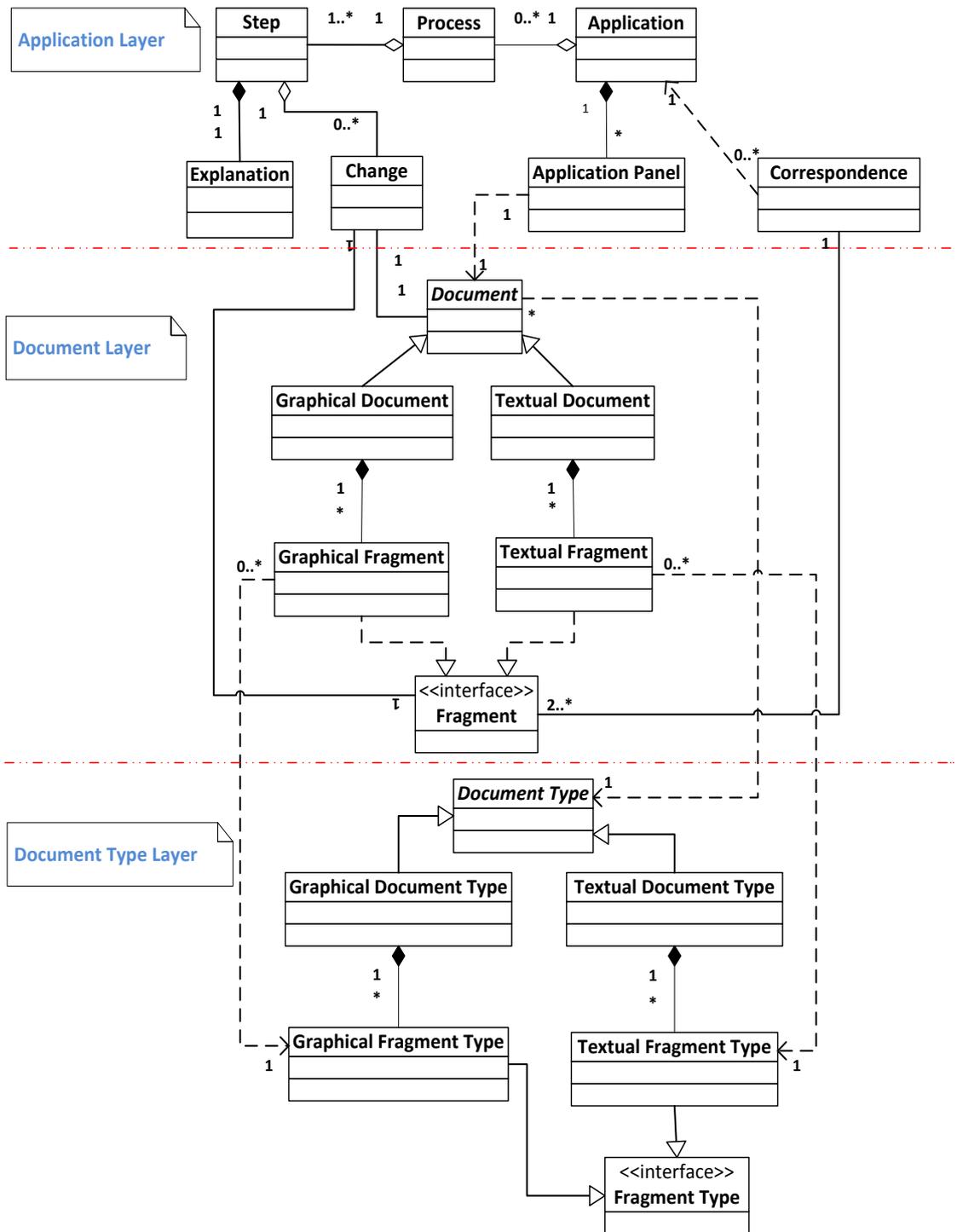


Figure A11.1 Class Diagram of IWE

Appendix 12 Extensible Markup Language (XML) Technology

As discussed in the requirement chapter, IWE requires saving the data of different parts worked examples. Hence, it requires choosing a data format to save this internal data into files. XML, which is a simple text-based format for representing structured information, was selected to achieve this task. There are several reasons for choosing XML to keep the data, which are:

XML is simple text; it can obviously be moved between various platforms. It is defined by World Web Consortium (W3C) which means it is a standard.

XML is one of the most widely-used formats for sharing structured information today: between programs, between people, between computers and people, both locally and across networks. XML allows a user to define a set of rules for encoding documents in a format, which customises the specification for representing data. The content of the document and the elements and attributes can be developed by the user. The content of document is both human-readable (by the developer) and machine-readable, but the users of IWE do not need to read these XML files.

The programming language that is used to develop IWE is Java and widely used XML parsers, such as SAX (Simple API for XML) and DOM (Document Object Model), can be used to parse a well-formed XML document into a typical data structure in Java.

The class diagram in Figure X.1 shows the complex low-level design of IWE; it requires the definition of custom specification XML to save data into different files. Using XML to maintain data can make low level implementations (coding work) easier.

XML provides a generic system that allows the definition of languages and the separation of “contents” from “presentation”. It meets the design requirement that displaying the knowledge contents as teacher’s required.

As described in the requirement chapter, the teacher must define and save: document types, documents, applications and processes. The initial version IWE keeps different data in different files, for example, keeping document types in a documentType.xml file, keeping document in document.xml and so on. Although this design is easy for the initial programming, it has the problem of synchronizing different xml files together. However, for the purpose of evaluating the research question, this design meets the requirement. Hence, following this design decision several separate XML files were created. The documentType.xml is defined and described as shown below.

DocumentType.xml

```
<?xml version="1.0" ?>
<doctypes>
  <fragmentType name = "name value" kind = "kind value" >
<awayfrom>away from value</awayfrom>
  <towards>towards value</towards>
  <nodestyle shape = "shape value" height = "height value" width = "width value" borderColor
= "border colour value" fillColor="fill colour value" labelColor="label colour value" ></nodestyle>
  </fragmentType>
  <documentType name = "name value" kind = "kind value" ID = "id value">
  <fragmentType name = "name value" kind = "kind value">
<from>from value</from>
  <to>to value</to>
  <arcstyle style = "style value" width = "width value" lineColour = "line colour value" labelColour
= "label colour value" ></ arcstyle >
  </fragmentType>
</documentType>
</doctypes>
```

Another big advantage of using XML file to save data is the benefit for user interface design. After the structure of an XML file is defined, the data that needs to be gathered from the user are identified. For example, in the documentType.xml file shown above, the red parts of xml file are the data required to be collected from the user interface. When designing the user interface, the only consideration is how to easily allow the user to input this required information. Details of the other different XML documents are presented below.

Document.xml

```
<?xml version="1.0" ?>
```

Appendix 12

```
<documents>
<document name = "name value" type = "type value" kind = "Graphical" FixOrder = "true" ID = "
">
  <node label = "label value" type = "type value" X = " " Y = " " ID = " " />
  <arc label = "label value" type = "type value" from = " " fromID = " " to = " " toID = " " fromIndex
= " " toIndex = " " ID = " " />
</document>
<document name = "name value" type = "type value" kind = "Textual" FixOrder = "true" ID = " ">
  <text type = "name value" value = "type value" order = " " ID = " " />
</document>
</documents>
```

Application.xml

```
<?xml version="1.0" ?>
<applications>
  <application name = "name value" layout = "1" appID = " ">
    <panels>
      <panel number = "0" type = "1" content = " " />
      <panel number = "1" type = "2" content = " " />
    </panels>
    <relations>
      <Connection id = "0" frags = "frag1_ID, frag2_ID,">
      </Connection>
      <Connection id = "1" frags = "frag3_ID, frag4_ID,">
      </Connection>
    </relations>
  </application>
</applications>
```

Processes.xml

```
<?xml version="1.0" ?>
<processes>
  <process name = "name value" app = "app name value" processID = " ">
    <step num = "1">
      <change num = "1">
        <docname>document name value</docname>
        <filename id = " ">fragment value</filename>
        <operation>Insert</operation>
      </change>
      <change num = "2">
        <docname>document name value</docname>
        <filename id = " ">fragment value</filename>
        <operation>Highlight</operation>
      </change>
    </step>
  </process>
</processes>
```

Feedbacks.xml

```
<?xml version="1.0" ?>
<Feedbacks>
  <Feedback userName = " " userSubject = " " type = "STUDENT">
    <Comment belongs = " ">Comment1</Comment>
    <Comment belongs = "">Comment2</Comment>
  </Feedback>
</Feedbacks>
```

Answer.xml

Appendix 12

```
<?xml version="1.0" ?>
<Answers>
<Answer userAnswer="" selectedOption=" " userName=" " subject=" " appName=" " procName="
" date=" " timecost=" ">
<question num = " " QuestionContent = " " type="MULTI_CHOICE">
<option num ="0" content=" "></option>
<option num ="1" content=" "></option>
<option num ="2" content=" "></option>
<option num ="3" content=" "></option>
</question>
</Answer>
<Answer userAnswer=" " selectedOption="-1" userName=" " subject=" " appName=" "
procName=" " date=" " timecost=" ">
<question num = " " QuestionContent=" " type="DESCRIPTIVE">
</question>
</Answer>
</Answers>
```

Appendix 13 System Usability Scale (SUS) Questionnaire and its Benchmark Lookup Table

System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	<input type="checkbox"/>				
	1	2	3	4	5
2. I found the system unnecessarily complex	<input type="checkbox"/>				
	1	2	3	4	5
3. I thought the system was easy to use	<input type="checkbox"/>				
	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	<input type="checkbox"/>				
	1	2	3	4	5
5. I found the various functions in this system were well integrated	<input type="checkbox"/>				
	1	2	3	4	5
6. I thought there was too much inconsistency in this system	<input type="checkbox"/>				
	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	<input type="checkbox"/>				
	1	2	3	4	5
8. I found the system very cumbersome to use	<input type="checkbox"/>				
	1	2	3	4	5
9. I felt very confident using the system	<input type="checkbox"/>				
	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	<input type="checkbox"/>				
	1	2	3	4	5

Scaled SUS Score	Percentile Rank
5	0.3%
10	0.4%
15	0.7%
20	1.0%
25	1.5%
30	2%
35	4%
40	6%
45	8%
50	13%
55	19%
60	29%
65	41%
66	44%
67	47%
68	50%
69	53%
70	56%
71	60%
72	63%
73	67%
74	70%
75	73%
76	77%
77	80%
78	83%
79	86%
80	88%
85	97%
90	99.8%
95	99.9999%

Table 1 Lookup Table for SUS Benchmark Data [97]

Sauro [1] published a global benchmark for SUS based on 446 studies (most have between 10 and 30 responses and some have more than 300) and over 5000 individual SUS responses, which combined the Bangor et al. dataset [2], Sauro 2011 datasets [3] and 129 studies from the Albert and Tullis datasets [4]. Sauro claimed that the weighted mean from all three sources provided an average SUS score of 68 with a standard deviation of 12.5. A look-up table between the raw SUS score and percentile rank was proposed, which is provided in Table 1. This percentile rank can tell how usable an evaluated application is relative to other products [1]. For example, an SUS score of 66 provides a percentile rank of 44%, which is considered more usable than 44% of all products tested using the SUS. Anything below 50% is below average and anything above 50% is above average, which means that the perceived usability in this example (66) is below average as 50% matched the SUS score of 68.

Reference

1. Sauro, J., A practical guide to the system usability scale: Background, benchmarks & best practices 2011: CreateSpace Independent Publishing Platform.
2. Bangor, A., P.T. Kortum, and J.T. Miller, An empirical evaluation of the system usability scale. *Intl. Journal of Human-Computer Interaction*, 2008. 24(6): p. 574-594.
3. Sauro, J. and J.R. Lewis. When designing usability questionnaires, does it hurt to be positive? in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2011. ACM.
4. Albert, W. and T. Tullis, *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics* 1st ed 2008: Morgan Kaufmann.