**Chapter 1**

**Introduction**

Teaching programming is inherently difficult. Literature on learning suggests that the most efficient way to learn a skill is by practice (reference). The way schools in the UK incorporate this approach is by explaining the basic concepts related to a specific topic, presenting simple examples to illustrate how these concepts can be applied and posing a more complex problem for pupils to solve. However, in the initial stages of becoming programmers, often beginners lack a good enough understanding of the domain to be able to solve the problem. This leads to pupils struggling to find a good solution, rather than gaining a better understanding of the problem-solving process.

A good way of teaching somebody an intellectual activity is by showing them the process of thinking involved in carrying it out This is a form of apprenticeship known in the literature as "cognitive apprenticeship"(Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. Educational Researcher, 18, 32-42). In any apprenticeship model, the learner needs to see many examples of the activity to be learned in order to develop the experience necessary to attempt a new, related, activity. Unfortunately, due to limited number of hours dedicated to each individual subject in schools, teachers are somewhat restricted to using only the traditional methods of teaching. Time simply wouldn’t allow them to show their pupils many examples of what cognitive steps they should undertake in order to solve a problem.

Research has shown that step-by-step guidance of the process of solving particular problems can help beginners gain a better understanding of the problem-solving process generally. Books provide such guidance in the form of worked examples that have proven to be effective. However, such books may not necessarily accommodate the needs of a particular teacher. Furthermore, finding a close enough example for a particular topic may become a time-consuming and discouraging activity for a teacher.

Having this in mind, a Glasgow University PhD student, Yulun Song, has developed a Java standalone application called *Interactive Worked Examples (IWE).* It aims to address the issues mentioned above as well as to evaluate to what extent such an application will prove effective in lowering the learning curve for pupils in Computing Science. It consists of two interfaces: one for students and one for creators of worked examples (who are typically teachers as well). The author interface enables the creation of examples to accommodate a teacher’s specific needs. The student interface provides users with a selection of examples to work on.

The application has proven to be effective at enhancing the teaching of Computing Science in university. Since the research questions around IWE were to explore the extent to which it can fit in the teaching process in a university context and whether it would be a potentially successful learning technique, the prototype does not aim at large scale deployment. A sensible next step is to put the system into use in schools, where support for computing education is urgently needed. However, many issues in deploying IWE arise because of it being in the form of a Java standalone application. In schools in the UK there tends to be a blanket policy about the systems provision on any subject. In order to install a program on a school machine, a request to the service provider responsible for the particular school needs to be made. The service provider will then need to analyse the risk that installing a new program will pose to the whole system and submit a further request to a local authority responsible for the particular school. This overhead would be enough to prevent most teachers from considering adoption, both from a time and cost standpoint.

The issue of software provisioning in schools gives the major motivation for this project to recreate IWE as a web-based application in order to start effectively presenting worked examples in a larger context. This will avoid the complicated and time-consuming process of installing IWE in schools. Furthermore, schools will be able to receive the latest updates of the application and its worked examples with no effort. Ultimately, a web-based system could share worked examples developed nationally and even internationally. The web-based version of IWE is called *Worked Examples Viewer (WEAVE)*.

In addition to being a more easily deployable version of IWE, WEAVE takes a step further to move from author-student to author-student-teacher target user groups. This brings in interesting new aspects. Teachers will be able to see personalised information about how their pupils interact with the examples. Authors, on the other hand, will receive information about the general usage of these examples, rather than personalised one.

Another benefit of WEAVE being web-based is that the worked examples in the system will not be limited to the ones created by one teacher or a group of teachers only. Instead, examples created by any teacher will immediately be available to everyone. This would contribute to a collaborative way of developing such examples and would give the chance for pupils to undertake further learning if they desired so. Furthermore, teachers would be able to benefit from their colleagues’ expertise as well as get ideas and adjust them to their specific needs with less effort than creating new examples from scratch. Ideally, such a system can be revolutionary in improving the teaching practices in schools, help teachers understand the difficulties of their pupils and enable them to help each other to become better in teaching Computing Science.

The rest of this dissertation describes more background for the context of the project, the requirements for, as well as the design and the implementation of WEAVE together with the testing methods that were used to ensure that the application works as intended. An evaluation chapter follows making conclusions about how easy and effective it is to integrate WEAVE successfully in everyday teaching practice. The final chapter is dedicated to the future developments for the system which will be addressed shortly.

**Chapter 2**

**Background**

**2.1. Worked examples**

**2.1.1. Definition of a Worked Example**

Clark defines a worked example as “a step-by-step demonstration of how to perform a task or how to solve a problem" (Clark, Nguyen, Sweller, 2006, p. 190). Another definition for worked examples is given by Atkinson as “instructional devices that provide an expert's problem solution for a learner to study.”(Learning from Examples: Instructional Principles from the Worked Examples Research). An effective worked example consists of a problem description, steps towards the solution and instructions at each step representing an expert’s process of thinking (Renkl, 2005). Of key importance is the step-by-step guidance for reaching the solution. It encourages the learner to form their own explanation for the undertaken step (Renkl et al, 2004) as well as think about what might follow next before they proceed. In essence, worked examples help novices to build an understanding of a concept so that in later stages they will be able to effectively apply to solve other problems related to this concept.

**2.1.2. Worked Examples and Learning**

The common assumption that the best learning is by practicing solving problems is not necessarily true for learning Computing Science. Renkl(2005) argues that without being exposed to worked examples first, novices have a very restricted knowledge on the domain to be able to effectively reach a solution. Solving problems involves a lot of working memory resources. However, the memory capacity of beginners should be used for building new knowledge instead. Clark argues that solving practice problems leads to using too much memory capacity thus not leaving enough of it for learning new knowledge (e-Learning and the Science of Instruction: Proven Guidelines for Consumers – Ruth C. Clark, Richard E. Mayer p.204).

Studying worked examples “is one of the earliest and probably the best known cognitive load reducing techniques” (Paas et al., 2003). It has proven to be effective in learning how to solve problems (van Merriënboer, 1997). While worked examples reduce the cognitive load, they also provide a better understanding of the concepts under consideration. This builds up the necessary expertise required to solve a particular type of problem effectively.

**2.1.3. Worked Examples in School Context**

The traditional methods of teaching Computing Science in schools across the UK do not include the best proven method to learn a cognitive skill described above. Often in schools, Computing Science concepts are introduced by explaining what the concept is, followed by a simple example. Then pupils are presented with a problem to solve themselves. The jump to problem solving is too quick and the importance of worked examples has not influenced the teaching methods. Keeping in mind that teachers are often limited time- and money-wise, a possible reason for this is because there is no easy means of finding and adapting existing worked examples to the specific needs of a teacher.

As part of his research project, the former Glasgow University PhD student Dr. Yulun Song developed software to facilitate the creation and viewing of worked examples. The thesis statement for the research outlines the basic aims for the project. The system developed is such that it:

* “delivers usable, best practice interactive worked examples to students in a computing science context;”
* “enables teachers to create such interactive worked examples without bespoke programming, and to evolve them on the basis of feedback from the students.”(reference)

Dr. Song was particularly interested in Computing Science problems due to their transformation-based nature. They involve the analysis and the transformation of one representation of the problem, such as text definition or a diagram, into another representation, i.e. the solution. An example described in the research thesis is building a database system from a specific set of requirements expressed in the form of a problem description in human language. The text describing the problem needs to be transformed into a graphical representation of the same problem - an ER diagram, which is then translated into a machine language such as SQL. Judgement and decision-making play a huge role in solving such a problem. However, these only come with experience and in order to gain such experience Reed & Bolstad (1991) claim that one example- which is the typical case in schools- might be insufficient. In his research, Dr. Song argues that a system that enables the user to view multiple worked examples would prove efficient in such a context. He therefore developed a tool for the provision of worked examples in Computing Science so that the user is exposed to more than one of those examples.

**2.1.4. Problems with Existing Methods for Delivering Worked Examples**

One can argue that worked examples can be found in many books and lectures so at first it may seem questionable what value would software bring to the existing provision. The thesis, however, raises some strong arguments to be taken into consideration.

* The worked examples in books or lectures are not interactive enough. The readers of books or the attendees of a lecture are presented with some examples, but often the process of thinking why a particular action is undertaken or is a better option for reaching a solution remains unexplained. One can argue that the university context has some grounds for interactivity or discussions. Yet many students may not exploit this due to shyness or simply because they might not know what questions to ask. Even if some interaction happens, this is not recorded or captured as part of the teaching process so the students cannot go back and review it.
* The worked examples present may not fit well enough to the teaching needs. Books aim to target a large portion of potential readers so they need to be general enough to fit every reader’s needs. However, this means that one particular reader may need to adjust their studying or teaching around this general example. What would be more beneficial - and Dr. Song aims to address - is to adjust the worked examples depending on the teaching or learning needed.
* Worked examples in books provide little or no feedback on how they were used to the author or to teachers who benefit from using such examples in their teaching. For example, the only available information for a book would be the number of copies sold. This would not provide any insight on the value the examples brought to the reader. What is desired is information about how a particular worked example was used, were there any problematic areas and how the readers benefited from it. Such information would allow the authors to improve their future work at constructing worked examples. In addition, this information could be beneficial to teachers or lecturers who could use them for assessing what parts of the example were problematic and adapting their teaching accordingly.

**2.1.5. How Does a Computer-Based Application Solve the Problems with the Traditional Methods of Delivering Worked Examples?**

The piece of software proposed and developed as part of Dr Song's work aims to address all the issues mentioned above. The student becomes actively involved with the material since revealing the steps required to reach a solution is under their control. The entire problem solving process can be fully captured and the students can easily go and review parts causing confusion. Complete explanation of every step is provided, enabling the student to follow the process of thinking of an expert. Revealing the solution step by step encourages thinking about the next logical step and guides the student towards the correct direction of thinking before they get confused. In addition, usage data can easily be captured to give feedback on how these worked examples were used. Data intended to be collected includes time spent at each step and answers to any questions present in the examples. Such information can be beneficial to two groups of people:

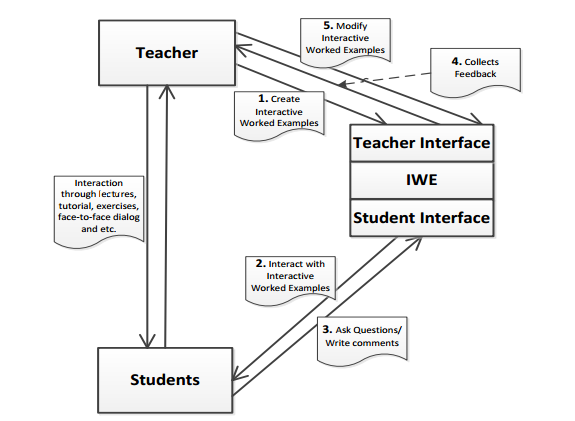
* Authors of worked examples. They could benefit from the knowledge of the time the majority of the students spend at each step. If this time exceeds dramatically the expected time for this step, this can be an indication that the step is unclear and brings confusion. Knowing this, authors could update the example by including a better explanation or by breaking this step into more than one steps and examine the effect this has. This way authors will learn how to build their examples and this will also bring benefits to the reader in terms of provision of improved worked examples.
* Teachers. The knowledge that their students visit a step multiple times or spend too long before proceeding would indicate to the teacher that their students do not understand the material for this step well enough and they might need to revisit it in class.

**2.2. Interactive Worked Examples Tool**

Dr. Song’s research product is called *Interactive Worked Examples (IWE)*. It is in the form of a Java standalone application. The following sections provide more information about the tool.

**2.2.1. Intended Flow of Interaction**

There are two well-distinguished groups of users – authors of examples and students. Each group is serviced by a separate interface of the application. The flow of interaction of these groups with the system, as presented in the thesis, is shown on Figure 2.1. The original figure can be found in Song’s thesis as Figure 2.6 (p.57).

Figure 2.1.

Authors can create and modify interactive worked examples through the author interface. This interface also presents them with any student feedback on these examples.

The student interface serves as a worked examples viewer where students are given the opportunity to ask questions and write comments.

The interaction between teachers and students is direct rather than through the system.

**2.2.2. Structure of IWE**

IWE stores the worked examples in XML files. There are three types of files that are of particular interest for this Level 4 project- Documents.xml, Applications.xml, and Processes.xml. The structure of those files is graphically represented on Figure 2.2.

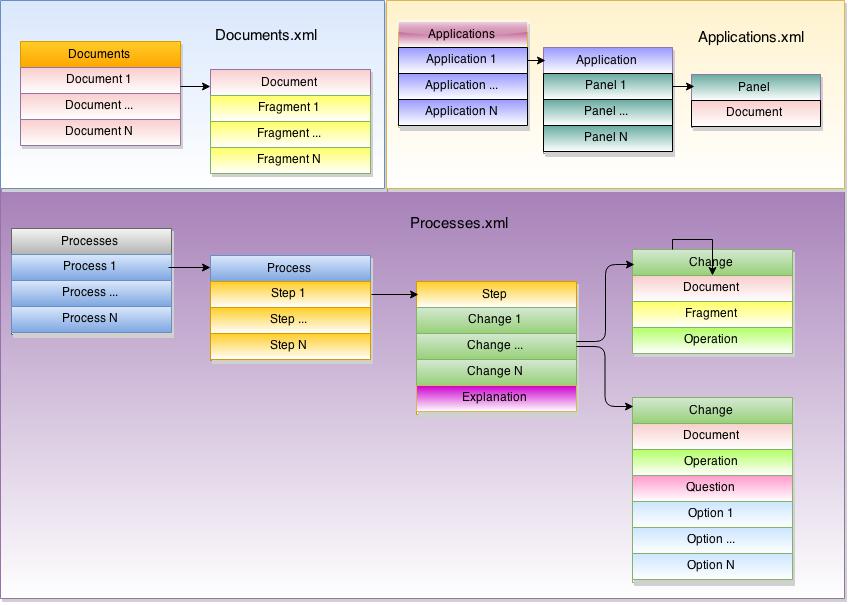


Figure 2.2.

The Documents.xml file stores the collection of documents created by an author. A document is one of the representations involved in a particular worked example- perhaps it is the problem specification, or an intermediate solution, or the solution. It is split into fragments which are small logically separated portions of the document. The reason for splitting the document into fragments is so that the document can be revealed gradually, to show the step-by-step problem solving process. Individual fragments can also be highlighted to be brought to the viewer’s attention.

The Applications.xml file stores layout information about worked examples, bringing together the particular documents involved in the worked example. The way the documents are laid out visually, in panels, is recorded.

The Processes.xml file defines the steps for the worked examples. For each step there are a number of changes and an explanation of those changes. There are two types of changes. The first type specifies which fragment of a document is involved in this change. These fragments can be shown, hidden or highlighted depending on the effect the author is aiming to achieve. The second type of changes corresponds to a question and possibly a set of options the user can select from in an attempt to answer it.

There is one more type of XML file which is not shown on Figure 2.2. It contains information about different styles that can be used for the worked examples- there is a similar, although simpler, version of the style mechanism found, for example, in word processors or CSS style sheets. There is no need this file to be discussed in any detail. However, the reader needs to know that documents have styles associated with them depending on the type of document, enabling different fragments within a document to be shown in different typographical styles.

**2.2.3. Main Features of IWE’s Student Interface**

The student interface aims to provide an effective worked examples viewer. A screenshot of the final version of Dr. Song’s prototype can be seen on Figure 2.3. The most important characteristics are labelled with numbers and are detailed below.

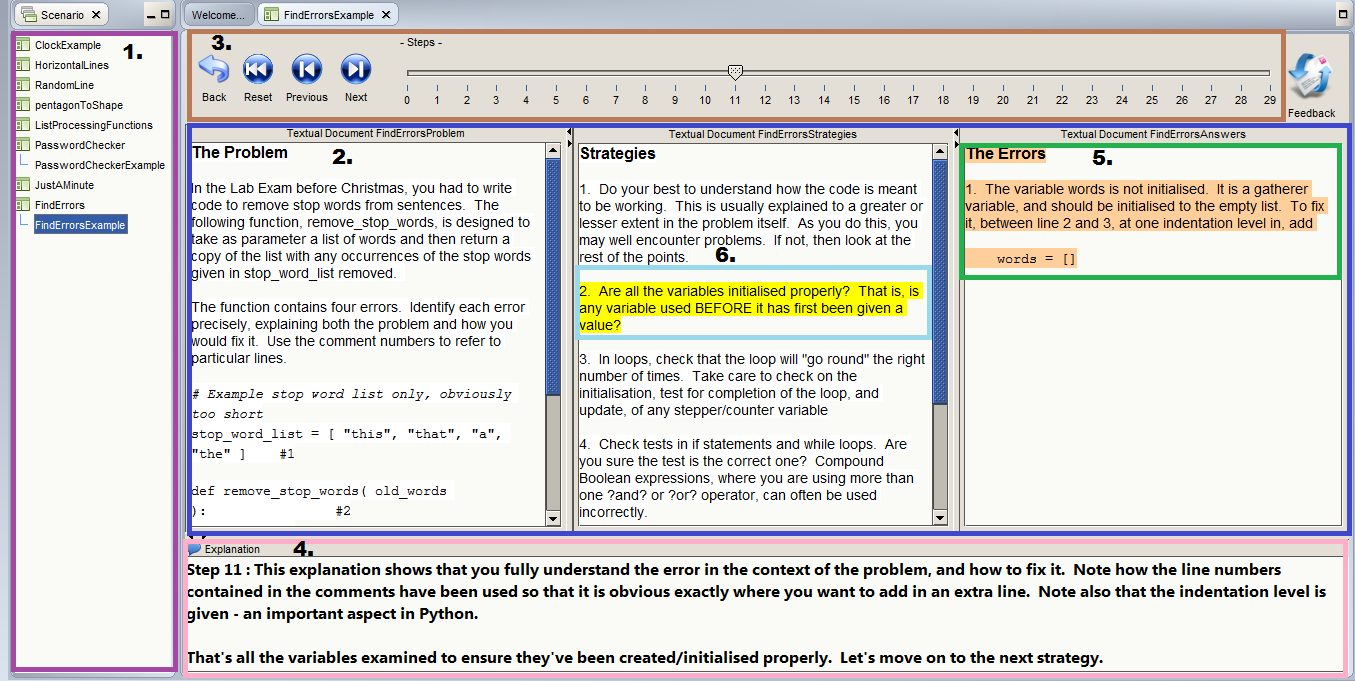


Figure 2.3.

1. An area for showing the worked examples installed on the system and enabling the user to choose an example to work on.

2. Panels showing different documents for a particular worked example.

3. An area for controlling transitions between steps.

4. An explanation area where the expert’s process of thinking involved on the current step is shown.

5. Highlighting of the newly appeared text at a particular step for drawing the user’s attention to the new content relevant for the current step.

6. Highlighting of fragments of interest for a particular step.

As the student uses the controls in area (3) to move through the worked example, the contents of the documents panels and the explanation area change to reveal the developing solution and the thinking process behind it.

Other features of IWE’s student interface, which are not shown on the screenshot, are the ability of the tool to ask the user questions and to record data such as time spent at each step and answers to questions.

**2.2.4. Relation of IWE to this Level 4 Project**

The evaluation on IWE conducted as part of Dr. Song’s research clearly shows the benefits of the tool. It has proven to be well-accepted and valuable as a technique to enhance a student’s learning experience. It also achieved its goal to enable teachers to more easily and quickly develop worked examples to fit their needs. The aim of this prototype, however, doesn’t cover deployment of the software in educational institutes but rather it has demonstrated that it would bring benefit to both students and teachers.

The motivation for this Level 4 project is to make use of the findings in Dr. Song’s research and take a step forward in deploying the software in schools across the UK. The focus of the project will be reconstructing the student interface of IWE into a web-based form, as well as providing a new interface for teachers to separate them as a distinct group of users. The author interface of IWE is beyond the scope of this project and no understanding of any of its aspects is needed by the reader to follow this dissertation.

For the rest of the dissertation, the reader’s attention will be drawn to the transition process of the Java standalone application IWE into a deployable online version called *Worked Examples Viewer (WEAVE)*.

# Chapter 3

# Requirements

This chapter provides a detailed description of the functional and non-functional requirements for WEAVE.

## 3.1. Background

As described in Chapter 2, this Level 4 project builds upon an existing system for facilitating the use of worked examples in educational context. The evaluation on IWE clearly shows that such software would be a valuable asset contributing to the learning process of students. Due to the overly complicated procedure required to deploy IWE in schools while it is in the form of a Java standalone application, the need to turn it into a more easily deployable online version arose. Interviews with highly motivated teachers, who are part of PLAN C project (reference), have identified the need for one more interface to be used in schools. In order to improve their teaching practice and to be able to provide high quality feedback to their pupils, these teachers would benefit from knowing how pupils in their classes use these worked examples. Information that would be valuable for them includes:

* identification of which pupils interacted with which examples
* aggregated information on answers selected for multiple choice questions and the pupils that selected each answer
* information about the average time spent at each step of an example as per the whole class
* information about the time spent at each step of an example as per an individual pupil of a class
* summary data of the total time spent at an example and the last step reached by each pupil in the class.

This project aims to achieve four goals:

**G1**- build a web-based viewing system that is interoperable with the author interface of IWE, i.e. ensure that worked examples created using the old system can be viewed in the new system.

**G2**- provide an interface for teachers that will help them gain more information on how the worked examples are used by their own pupils.

**G3**- replicate as closely as possible the student interface of IWE.

**G4**- ensure that worked example authors can view usage data in an anonymous manner, such that individual pupils, classes or schools are not identifiable.

## 3.2. Classification of Requirements

The requirements are classified according to the *MoSCoW* classification method (reference). The categories considered are:

- **must-have**- requirements that are crucial for the achievement of the goal of this project and must be implemented

- **should-have**- requirements that are considered to be important but not necessarily crucial for achieving the goal of this project and should be implemented

- **could have**- requirements that have been identified as features that would add further value to the prototype but are thought of as stand-out ones rather than ones contributing to the correct functioning of the prototype and may not be implemented due to constraints

The **would-like** category coming from the **W** in MoSCoW is not part of the classification methods used for this project due to the fact that all the requirements fit comfortably in the other categories.

**3.3. Functional Requirements**

**3.3.1. Interoperability with the existing author interface**

The existing system uses XML files to store the worked examples. The structure of these files is shown on **Figure 2.2** in the Background chapter. The web-based system will need to read in worked examples from these data files. Furthermore, feedback from pupils and teachers will inevitably lead to changes being required in some of the worked examples. The existing authoring tool supports editing of worked examples, and it is expected that it will still be used to make such changes. WEAVE will need to be able to support these changes. Due to the fact that the update model of IWE is destructive- no versioning of the examples is supported- and that WEAVE does not provide means for modifying examples, the update model will follow the one of the old system.

The prototype:

* **must** be able to parse an XML file containing the fragmented problem specifications of the worked examples and their solutions.
* **must** be able to parse an XML document containing information about individual steps of the worked examples (e.g. which fragments of a document must be shown/hidden/highlighted, the explanation associated with a step and a question if one was provided).
* **must** be able to parse an XML document containing information about the layout of worked examples (e.g. number of panels for the example, their order and problem solutions associated with each panel).
* **must** be able to parse an XML document containing information about the styling associated with each example (e.g. font style, font size, etc.).
* **must** be able to support easy addition of new worked examples created using the old authoring tool.
* **must** be able to incorporate new versions of worked examples already installed in the web-based system.

**3.3.2. Teacher interface requirements**

A major part of the contribution of WEAVE is to enable teachers to receive information about how their pupils worked with these examples, while authors of such examples and Computing Science researchers must receive such data in an anonymised way. The desired effect is teachers to be able to see usage data for their classes as well as individuals in these classes. However, protecting the privacy of both teachers and pupils is a major issue. The authors of worked examples will be able to see any usage data for the examples they created. If this data is informative enough for them to identify the person standing behind this data, this would be highly unethical and would violate somebody’s privacy.

In this section, the requirements for the teacher interface are outlined. The next chapter will describe how the privacy issues mentioned above are resolved by the system and will discuss in further detail these requirements.

The teacher:

* **must** be able to register with a username and password.
* **must** be able to login/logout of the system.
* **must** be able to create groups for their pupils
  + **must** be able to specify the name of the group.
  + **must** be able to uniquely identify the pupils in each group
  + **should** be able to specify the number of pupils for the group.
  + **could** be able to specify the academic year this group belongs to.
* **should** be able to update existing groups by adding more pupils to them.
* **should** be able to view a printable list showing the pupil ids for a group.
* **should** be able to view information on the average time spent by all pupils at each step.
* **should** be able to view information on the number of times an answer for a question has been chosen.
* **could** be able to view information on the average time spent by a particular pupil at each step.
* **could** be able to view information on the list of pupils that chose a particular answer to a question.
* **could** be able to view information on the total time a pupil spent on an example.
* **could** be able to view information on the last step a pupil reached on an example .
* **could** be able to delete existing group.

**3.3.3. Replication of the IWE Student Interface**

One of the goals for this Level 4 project (**G3**) is to replicate the student interface of IWE as closely as possible. The reasoning behind the requirements for the student interface as well as the positive conclusion from their evaluation are described in detail in Dr. Song’s thesis. These were found acceptable for the purpose of this project.

The prototype:

* **must** enable the pupil to select a worked example from a list of existing examples.
* **must** support multiple panels for the different parts of the problem solution.
* **must** contain a dedicated area for the explanation.
* **must** support showing/hiding/highlighting of fragments.
* **must** support the option to ask pupils questions.
* **must** enable the pupil to go back and forwards through steps.
* **should** record time spent at a step.
* **should** record answers to questions
* **should** enable the pupil to reset the example there are working on.
* **should** highlight the newly introduced fragments at each step.
* **could** provide a means for drawing the pupil’s attention to the newly introduced fragments.

**3.3.4. Additional Features Needed for the Student Interface**

This section describes the requirements for satisfying goals **G2** and **G4** – supporting identifiable usage information for the teacher and anonymous usage information for worked examples authors and for Computing Science education researchers. It is important that each teacher is able to link usage data to their groups/ pupils while authors of such examples must not be able to identify by any means what the group or who the pupil is due to the privacy issues discussed above.

The prototype:

* **must** be able to show personalised usage data to the teacher.
* **must**  be able to show anonymous usage data to authors of worked examples and Computing Science education researchers.
* **must** allow the pupil to use the system without any identifying information.
* **must** be able to connect the usage information stored for a pupil to their teacher.
* **must** be able to connect the usage information stored for a pupil to their teacher and the current academic year.
* **must** be able to connect the usage information stored for a pupil to their teacher, the current academic year and a group they were allocated to.
* **must** connect the usage information stored for a pupil to their teacher, the current academic year, a group they were allocated to and a pupil id.

# 3.4. Non-functional requirements

The non-functional requirements for WEAVE are guided mostly by the web-based nature of the system and by the context it is intended for. In order for pupils to be able to study the worked examples effectively, and also due to the small workstation screen sizes found in schools, the area showing the worked example should be maximised. Furthermore, due to the step-by-step nature of worked examples, some steps may put more emphasis on the explanation while others might be more intensive in the problem specification areas so the system must be able to deal with such situations accordingly. In addition, features which make the interactions with the examples more convenient and which would minimise effort, such as shortcuts and appropriate fitting of the whole system on the screen, are highly desirable.

The web-based nature of WEAVE poses a possible problem when uploading modifications to existing examples because pupils might be working on these examples at the same time. Consistency must be ensured in such cases, meaning that the pupil should be able to see either the old or the new version of the example, rather than a mixture of both.

Since pupils may have not worked with such a system before, they may benefit from a brief guide on how to use WEAVE in an optimal way.

These considerations form the following requirements:

* The prototype **must** be easy to use.
* The worked examples **must** fit the entire screen.
* The size of the area showing the worked examples **must** not exceed the size of the screen.
* A modification to a worked example **must** not affect pupils doing the same example.
* The student interface **should** include a tutorial on how to use the system
* The teacher interface **should** provide information on how to use each feature.
* The panels showing the problem content **should** be resizable.
* The explanation area **should** be resizable.
* Shortcuts for easier transition between steps **could** be added.

The following chapter will describe the design decisions which were constructed based on these requirements.

**Chapter 4**

**Design**

This chapter outlines the design decisions for this Level 4 project. As it was previously described, WEAVE is based on top of the standalone Java application IWE. This affects to a great extent these decisions. Following the well-established software engineering principle of reusability, guidance for the design of WEAVE was to reuse any good aspects of IWE’s design while improving its weaknesses.

**4.1. Storage of Data**

The scope of this Level 4 project does not include the creation of a web based author interface. This means that the design of WEAVE must ensure an easy and efficient way for importing and storing the existing worked examples. In addition, WEAVE must support uploading updates to existing examples. As described in **Chapter 2**, the examples are stored in the form of XML files. To remind yourself of the structure of these files, please refer to **Figure 2.2**.

During the exploration process of IWE, a weakness of storing the information about the examples in the XML files used by IWE was identified. Since these files are easy to access and modify by the authors of worked examples, and this is typically much easier than modifying the worked examples using the author interface, one may be tempted to make changes to the examples manually. However, references to some objects may be present in more than one of these XML files. For example, in the Documents.xml file fragments are stored as individual elements identified by a fragment id. When the steps for the example are defined in the Processes.xml file, each fragment is referred to with its id. A problem with storing information about the examples in files is that if an element is modified in one of the files, consistency about this element must be ensured. It is trivial having to find the same feature across multiple files and in the end consistency and validity are not guaranteed.

Django provides object relational model (ORM) database functionality. This is an efficient way to manage objects and their relationships and is a preferred method for managing the data for the worked examples because it guarantees consistency. The relationships between different objects are expressed via foreign keys. The entity relationship diagram is presented on Figure 4.1. Using a database adds a level of reliability that the data is valid because internal integrity checks are made before saving an object to the database. Further advantage is that the Django administrator interface allows an easy means of exploring and modifying the examples in a consistent and safe manner.

**4.2. Authentication**

A huge discussion point was how to authenticate teachers and their pupils due to the privacy and ethical issues discussed in **Section 3.3.2**.

One possibility was pupils to create their own accounts and give their usernames to the teacher. However, this approach could potentially result in various complications.

* Pupils would need to be explicitly instructed that their accounts should not reveal their true identity.
* Pupils would not have any benefit of having their own accounts and only the teacher will be the one who would use their usernames for something meaningful, i.e. to check their progress.
* This approach solves identification at an individual level but each pupil needs to belong to a group as well.

A second option was considered, in which the teacher would create an account for each of their classes. For this account, they would need to create usernames for their pupils. Keeping in mind that a teacher would often have more than one classes and that each class consists of twenty to thirty pupils, the following problems arise:

* The teacher would need to create a lot of accounts and this could be a trivial and time consuming task.
* The teacher would need to ensure they will be able to match each of their pupils to their id since they must not use any names due to the privacy issues mentioned above.

To go around the privacy issues and the problems with the options described above, a third approach based on the general idea of the second one was adopted. Teachers will create their own accounts. In these accounts, they will be able to create groups for their classes. On creation of the group, the teacher needs to specify the number of the pupils in this class. WEAVE will then generate random ids for these pupils. An id consists of two random letters followed by a single digit. While being short enough to be easily remembered, such an id ensures that the privacy of pupils is protected due to its random nature. Furthermore, the number of possible combinations for all pupil ids is large enough so that it will be highly unlikely that pupils will be able to “guess” one of their classmate’s pupil id and work with the examples on their behalf.

Talking to a lead teacher, Peter Donaldson, who is part of the PLAN C project, a potential inconvenience of this approach was identified. Using WEAVE for longer periods than one academic year could lead to a significant increase of the groups. In addition to the growth of the number of groups, some teachers might prefer to use the same name for their classes across years. To resolve these issues, a further classification of groups by the academic year the group belongs to was adopted.

Mr. Donaldson was generally happy with the idea that teachers select the number of pupils for each group at the creation of this group. However, he pointed out that it is possible for a pupil to arrive in a class at a later stage than the beginning of the academic year. Using the selected approach would have required that the teacher creates a new group just to add one pupil only. This could be very problematic, because data about the same pupils would be spread across two different groups and most of the pupils will be given two pupil ids which could become really confusing for both teachers and pupils. To avoid these problems, the option for teachers to update a group was added to the design decisions for the authentication part. In addition, groups can also be deleted in case of creation of unneeded groups.

**4.3. Presentation of Data**

The main goal of the teacher interface is to present to the teacher data associated with a particular group or pupil. Three different types of data are recorded from the student interface:

* Time at each step.
* The direction of the transition to each step i.e. is the pupil going backwards or forwards to a step.
* Answers to questions.

Careful consideration was needed to reach to a solution that would visualise this data in a way which would enable the teacher to comprehend it easily and encourage further analysis. In addition, the ability the teacher to be able to view data both at a class and at an individual level further influenced the design decisions.

To start with, it has been decided that presenting the data in the form of graphs would be hugely beneficial to teachers as they would be able to identify patterns and any exceptional events for a particular worked example. These graphs should reveal information about the performance of the whole class as well as of individual pupils at each step of the example keeping in mind that some steps have questions. For this purpose, five different types of graphs were decided upon:

* Average Time. This graph would show the average performance of the pupils in the whole class on a particular example. Different steps will be represented on the x-axis of the graph by their step number. The average time spent on each step will be shown by the y-axis. Since the x-axis consists of the step number, this is not very informative to the teacher because they would need to look at the actual example to remind themselves about the context of the step. To avoid the need for that, hovering over the point representing the step will show the beginning of the explanation. Clicking on that point will open a window showing the whole text of the explanation, the average time spent on that step and how many pupils made a backwards transition to the step. As mentioned above, some steps contain a question, rather than an explanation. Such steps will be identified by a question mark in front of the step number on the axis label. Instead of showing an explanation on mouse hovering, a message encouraging teachers to click in order to see pupils’ answers is shown. Clicking on the point for that step will show the question, all the possible answers and how many and which pupils selected each option.
* Pupil Time. This graph is conceptually the same as the Average Time graph. Instead of showing information about the whole class, however, it shows the total amount of time spent at each step by a selected pupil.
* Pupil Answers. This graph shows the options for a selected question and the number of pupils who chose each option. Hovering with the mouse over each bar shows the list of pupils who selected the answer represented by this bar.
* Class Steps. This graph shows information about the time spent at a chosen step of an example by the pupils of a class. It is in the form of a bar chart where each bar represents a pupil’s attempt. This means that there might be more than one bar for each pupil if they have attempted the selected step more than once- each bar revealing information about the time spent by that pupil at a particular attempt of the step.
* Class Summary. This is a table showing summary information about the total time spent by each pupil at a particular example, how many times they returned to previous steps and the last step they reached. This would show the teacher how much effort did the pupil put in each example, how many problematic or unclear steps they encountered as well as whether they completed the example or which step they gave up on.

**4.4. Architecture**

The architecture of Weave consists of three distinct tiers as visualised on Figure 4.2.

* Presentation tier. This is the top level of the overall architecture also known as the client side web interface. It defines the appearance of the website by rendering HTML and CSS and provides means for users to interact with the application. The clients are in the form of web browsers. On every interaction, they send requests to the server in the form of Ajax GET or POST requests to ensure asynchronous communication between the client and the server.
* Django Middleware. This tier consists of two distinct components.
  + The first component of this tier serves as a communication point between the client and the database. In this tier the requests from the client are parsed and translated into ORM requests- a language understandable by the database. These requests are passed forward to get or store the information in the request from/in the database. After the backend generates the response, the middleware is responsible for passing it back to the presentation tier in the form of Http response.
  + The second component is the connection point between IWE and WEAVE. This is where the translation of the XML elements storing the examples in the form of ORM objects is taking place. An XML parser is used to separate the elements created by the author interface. The relationships between these elements are established and the database is populated with the objects defined by the parser.
* Data layer. This tier represents the database in which all the information used or generated by WEAVE is stored in the form of objects. On GET requests the backend tier responds with an object meeting the criteria specified in the ORM request. On POST requests, the database creates a new object with the features specified in the request and stores it in the database.

**4.5. User interface**

As already mentioned, WEAVE is intended for three distinct groups of users, each with their different needs. However, for the purpose of this Level 4 project, only two of these groups will influence the user interface. Addressing the needs of authors is beyond the scope of this project. To add examples to the system, they need to use the IWE author interface and contact the administrators of WEAVE with a request to add the newly created example to the application. This is why the section on the user interface is split into two subsections only, which describe the user interfaces for pupils and for teachers.

**4.5.1. Student User Interface**

A core purpose of this Level 4 project is to translate the student part of IWE into a more easily deployable online version. Careful consideration about the layout of the student interface of IWE is evident. The evaluation of IWE proved that the current interface is well accepted by students. A screenshot of this interface is presented on **Figure 2.3**. Generally, it has been decided to take advantage of Dr. Song’s findings and to reuse a very similar interface.

**4.5.1.1. Home page**

The need for some additional features of this interface arises due to the fact that teachers need to be able to identify their pupils in order to monitor their interaction with the examples. To ensure that pupils are using the system in the intended way and to encourage them to use the details provided by their teacher, it has been decided that the examples will be hidden to them until they enter some authentication information or identify themselves as anonymous users.

Once the pupil has identified themselves, the area prompting the user for details is exchanged for the list of worked examples. Due to the fact that the system is required to accommodate examples created by many teachers across the UK, a filtering by the name of the worked examples functionality has been provided. In addition, the option to select a worked example appears in the top navigation bar. Having the same feature twice might seem repetitive at first. However, more careful consideration justifies this design decision. Substituting the detail specification area with the list of examples guides the user that they need to choose an example to work on. Having the same list of examples in the toolbar area contributes for faster navigation between examples.

In order to be able to exploit the worked examples viewer in an optimal way and to familiarise pupils with it, a tutorial appears on the main page. The idea for having a tutorial was borrowed from IWE. However, the way this tutorial was constructed there was identified as potentially ineffective at communicating all the information the pupil needs to know before working on examples due to the fact that it contains a lot of text which may discourage some of the pupils to read it. Furthermore, even if they read the tutorial, they may not understand what is referred to in the text because they may have not seen the worked examples viewer and its features in advance. A different approach was chosen for the tutorial on WEAVE. It is split into different steps describing an individual feature using minimal text and a screenshot of the feature.

**4.5.1.2. Page for viewing an example**

The page for viewing a worked example is very similar to the one used for the IWE student interface. However, due to constraints imposed by the size of the screens in schools, the design needed to be adjusted accordingly. The area for selecting an example (referred to as **Element 1** in **Figure 2.3**) is placed on the navigation bar with all the examples appearing in a drop down menu on request. This saves a significant portion of the screen which can be used for the problem specification instead. Another space consuming element is the bar showing the current step. In WEAVE this information is shown as a part of the explanation instead.

**4.5.2. Teacher Interface**

**4.5.2.1. Home page**

**4.5.2.1.1. Non-logged teachers**

The purpose of the teacher interface requires the teacher to be logged in. Therefore, on the first visit of the page the teachers are presented with a register and log in areas only.

**4.5.2.1.2. Logged Teachers**

The discussion on the design decisions on the authentication to WEAVE and the way usage data is presented to the teachers identifies the main sections of the interface for the logged in teacher. Options for all the activities a teacher can undertake via the teacher interface are present on their home page to avoid the need for transitions between different pages and to simplify navigation of the website. The main page is split into three areas.

* Area for registering, updating and deleting a group. These three options are provided in the same area on the screen. When the teacher selects the desired option, the elements for this area change accordingly. For example, when the user wants to create a group, they need to enter the group name and the number of pupils for that group. On update or deletion of a group, on the other hand, they select the group name from a dropdown list. The list of existing groups is shown to remind which group names are unavailable to this teacher. The textbox for entering the number of pupils accepts integer input only for error prevention purposes. On the submission of the request to create/update/delete a group, a message confirming the status of the action is show.
* View Group area enabling the teacher to select a group for which to view the pupil ids. Again, for simplicity and error prevention, the teacher selects the group via a dropdown list rather than typing its name.
* View Statistics icon which navigates the teacher to the statistics page.

**4.5.2.2. View Group Page**

Teachers are able to see the pupil ids for a class in the View Group page. This information is provided in a table form with columns for the pupil name and two identical columns with the pupil id. Teachers will be advised to print this group sheet and fill in the names of their pupils by hand. This would avoid any potential problems with storing identification information in the system. Teachers will also be encouraged to cut one of the columns for the pupil ids and hand them privately to each pupil.

**4.5.2.3. View Statistics Page**

Most of the design decisions for the different types of graphs are explained in **Section 4.3.** above. The teacher needs to select the particular group and the type of data they are interested in. If there is no data for that selection or the selection is invalid, an appropriate message appears on the screen. Otherwise, a graph is shown. This graph is downloadable to enable saving the data for statistics at different points in time and could be used for comparison by the teacher.

The following chapter will describe how the design decisions for WEAVE were implemented.