

# IMPLEMENTING IMMERSIVE LEARNING ENVIRONMENTS FOR TEACHING SOFTWARE ENGINEERING

Richard Cooper  
Computing Science  
University of Glasgow  
Glasgow G12 8QQ  
rich@dcs.gla.ac.uk  
<http://www.dcs.gla.ac.uk/~rich>

Can Wang  
Computing Science  
University of Glasgow  
Glasgow G12 8QQ

---

## ABSTRACT

*Many of the concepts that need to be taught in software engineering courses are hard to get across because it is difficult to give the student practical experience of the intuitions and techniques involved. The student must come to understand multiple design and implementation formalisms whose relationship is often hidden by modern commercial software. Education theory would suggest that students would learn better if they were given practical experience of these hidden concepts. We have previously described the design of a framework which permits immersive environments to be constructed easily. This paper extends the design and describes the implementation of a prototype framework.*

## Keywords

*Immersive Learning, Software Engineering, Database Teaching.*

## 1. INTRODUCTION

The teaching of the various aspects of software engineering is problematic since the intention is to get the student to use unfamiliar ways of thinking and modes of expression to describe information structures and processes precisely. Previously [1], we have described how the variety of formalisms to be mastered, the unfamiliarity of their concepts and the obscure nature of the inter-relationships between different formalisms has made teaching more complicated. We also noted that the problem was further exacerbated by two extra features of the development cycle. Firstly, some tasks are

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

© 2006 Higher Education Academy  
Subject Centre for Information and Computer Sciences

accomplished off-line and consequently, giving feedback to errors is slow. Secondly, when giving the student practical experience of commercial software, many of the processes are achieved automatically and therefore remain hidden and not accessible to observation.

We further noted that educational theories increasingly emphasise the value of having students enter into a conversation with their educators concerning the material they are trying to learn [2] and that they should interact actively with the material [3]. The use of computer-assisted learning has long been proposed as a means of bridging the gap [4], in particular by creating a simulated environment in which a student can test out a theory by exploring the effects of different inputs or by varying parameters. Such an environment actively involves the student in working with the material and, if suitably constructed, provides feedback which makes the use of the environment conversational.

We proposed, as a solution, environments in which the student can interact with the whole space of application description from requirements to implementation. The student could then become immersed in the environment and could then view and manipulate the variety of representations available. The requirements of such an environment are that representations at different levels of abstraction should be available, that the correspondences between different representations be made apparent, that the processes involved be capable of being visualised and controlled and that explanatory comment be consistently available to guide the student in their exploration.

One way of achieving this is by creating a collection of programs, each allowing the student to experience one aspect of software development. There are two problems with this approach - each program would require separate effort and there would be a lack of continuity between the different programs. We proposed instead a framework for

developing and integrating these programs, which exploits firstly the fact that each program uses largely the same techniques and secondly that they can interact with a common underlying set of data structures.

This paper revisits some of the motivation, but then continues with the practical aspects of producing such an approach, culminating in the description of a first small prototype.

## 2. SOFTWARE ENGINEERING CONCEPTS

The software engineering part of computing courses comprise modules which teach the techniques of programming and database construction, but also the capture of requirements, the management of projects, maintenance techniques and the methodology used. Our previous paper [1] took a high level view of what is being taught, grouping the concepts into a number of areas. We summarise those areas here:

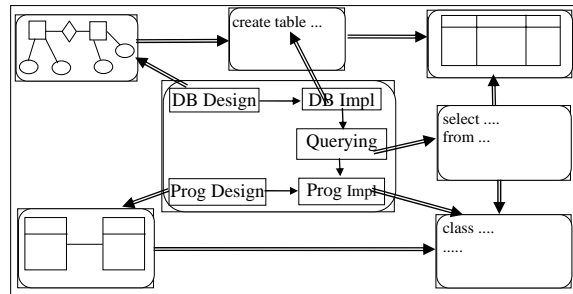
- Most importantly, there are the half dozen or so building blocks of programs (loops, conditionals, etc.) and a similar number of building blocks of data structures (sets, booleans, etc.).
- Then there is the difficult activity of extracting a set of *usable requirements* for the product.
- The student must then master a number of design-level languages such as ER and UML diagrams.
- Much of the more concrete (and problematic) tasks of teaching in computing concerns helping the student to gain mastery of implementation-level languages, including programming, database and mark-up languages.
- Exposure to commercial software and its practice is another component of a module and there may well arise a gap between the student's understanding of theory and practice due to the way in which commercial software simplifies tasks for its users.
- All of these may also be embedded in a software engineering methodology, such as extreme programming.

So the structures of the essential ideas of software engineering are few and we can identify the following abilities:

- to write down formally a specification of a piece of software or a piece of data given the description of some real world activity or information;
- to map one representation of software or data into another;
- to step through a process;

- to understand how components from two representations are equivalent; and
- to relate practice to the taught concepts.

We therefore envisage a complete environment in which all of the structures can be described and related and the processes they go through be open to investigation. At any time a description can be entered and validated, then transformed into another description. Where two equivalent descriptions exist, correspondences between elements of each can be identified. Each object can also be related to description.



**Figure 1. A Teaching Environment for Software Engineering**

Figure 1 shows how an immersive teaching environment can be organised. The centre icon shows a methodology diagram, each icon of which leads to a frame in which a particular representation can be edited. The design specifications also lead to the implementation specifications and querying can be tested and also embedded in a program, using techniques such as JDBC. Data can also be visualised. All representations are kept consistent, so that editing one is reflected in changes of any equivalent representation.

## 3. COMPONENTS OF IMMERSIVE ENVIRONMENTS

The kind of program which we envisage is one in which the student is provided with tools for entering formal documents, such as programs or design diagrams, and then is allowed to experiment with the effect of the document – either by observing the effect of execution or of mapping to a different formalism. Documents may be textual or graphical and may be at the design level or implementation level.

In all cases, the student is expected to enter the document either through the use of an editor or by loading a file prepared previously. The document is composed of fragments, each representing one unit of software construction. Fragments may be variable names, keywords, data values, etc. The document will be displayed in ways which highlight the differing nature of the fragments. Colouring text or using different shapes for icons are two obvious

ways of achieving this. Documents are displayed in panels in the window and fragments can be selected for modification or removal.

In order to create a framework for building applications, we therefore need to be able to be able to describe, on the one hand, a set of structures for describing the software being constructed and, on the other, to be able to describe the user interface which gives the student access to the software engineering processes. The first of these leads to the following set of requirements for describing the software structures available for manipulation in the immersive environment:

- the ability to describe documents, both graphical and textual, each of which is made up of distinguishable and typed fragments, each of which is separately manipulable;
- the ability to describe a set of underlying data structures (to describe software objects not the data which the software manipulates) so that the document fragments can be mapped onto values which can be stored;
- the ability to describe correspondences between documents in which fragments of one document is mapped onto fragments of the other – e.g. mapping ER entity types to tables;
- the ability to describe processes over the data which are composed of atomic actions such as add a fragment, highlight a fragment and so on.

The user interface requirements are as follows:

- the ability to describe a multi-frame application with each frame consisting of multiple panels – each panel will be called a region;;
- the ability to create components for displaying the textual and graphical documents and editors for the documents;
- the ability to allocate documents to regions in the interface;
- the ability to step through processes – either the processes of transformation or the processes that the designed software is supposed to be supporting;
- the ability to describe tutorials, glossaries and help systems which can be linked to the rest of the programs;
- the ability to describe menus, toolbars and so on.

The next section describes our model for achieving these requirements.

## 4. A MODEL OF SOFTWARE ENGINEERING TEACHING TOOLS

The framework supports the creation, editing and relating of applications which manage software engineering documents. Here are informal definitions of the main structures managed by the framework. The model comprises two parts - one for the information structures and the other for the user interface structures.

### 4.1 The Structure of SE Documents

A *document* is a collection of *fragments*. A document has a *type* and the type specifies a grammar to which the document must accord. Fragments come in a variety of kinds and these are best described separately for textual and graphical documents.

Textual fragments can be keywords, metadata constructs names, metadata names and data values. Thus in:

```
select name from person where age = 25;
```

select, from and where are keywords; name, person and age are metadata names and 25 is a data value. The use of the word “table” in SQL *create table* statements is an example of a metadata construct name. Graphical fragments can be icons, text or lines. Fragment types can also be made to be capable of being hyperlinks – thus every keyword could link to a glossary definition.

A document type has a *grammar* for the way in which the fragments are structured. The grammar defines the specific fragment types which can appear. It also limits the sequence of textual fragment or the ways in which graphical fragments are linked.

The grammar is rooted in an underlying *data structure*. Thus the grammar for the *select* statement above is rooted in a data structure capturing the relational model. The constructs of the data structure are described first and then the grammar of the document type is defined in terms of the data structure constructs.

Fragments between different document types are linked by *correspondences*. For instance, an attribute in an ER diagram is linked to a column in a relational data definition document.

A *process* is a sequence of actions on documents. Thus the mapping of an ER diagram to a set of create table statements is a process which starts by starting with an ER document and taking each entity type in turn and adding a line of the form

```
create table <entityType> ()
```

to a relational create table document. Then each other part of the ER diagram is taken in turn and

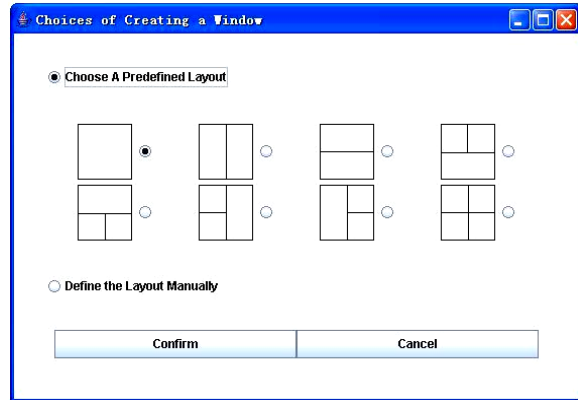
used to flesh out the create table document with column names and so on.

Particular document types which are not specific to any particular software engineering technique are also provided. For instance, a feedback window is a window which allows messages to be displayed (either by being appended or by replacing the previous message). A glossary is a list of definitions, while a help system is an organised set of instructions indexed by keywords.

## 4.2 Describing the User interface

The user interface description starts by defining the interface as a collection of *windows* and *regions* within windows. It continues with an association of each region with a document type and a *manager* for that document. The manager may be a display module or an editor for documents of that type. A display module supports the operations of adding fragments to the document. An editor also allows operations such as deletion, modification or relocation. Textual display modules may use hyperlinks to relate them. For instance, the feedback window might include hyperlinks to the glossary.

Document fragment types are associated with *styles*. The style of a text fragment type indicates its colour, font, font size, etc., while the style of a graphical icon indicates its colour, shape, and so on.



Processes are managed by *progress controls*. These allow a process to be executed either automatically or by student-controlled step functions. A cassette player style panel of controls is one mechanism for this.

## 5. A PROTOTYPE FRAMEWORK

A prototype system to enable the construction of very simple immersive tools has been produced. The prototype has been written in Java which conveys significant benefit. The Swing tools for creating windows, menus, toolbars, etc. is of enormous benefit and the Swing Text Framework supports the creation of hyperlinked and styled documents of precisely the kinds we need.

Creating an application with the tool begins by designing a window layout as shown in Figure 2.

The next step is to create a grammar for each document type as a text file which can be imported. Then each fragment type is associated with a style. This process is shown in Figure 3. At the top is panels for listing and creating styles, while at the bottom we see how styles are associated with keywords. A similar interface is used for graphical fragment types.

Figure 2, Creating a Window Layout

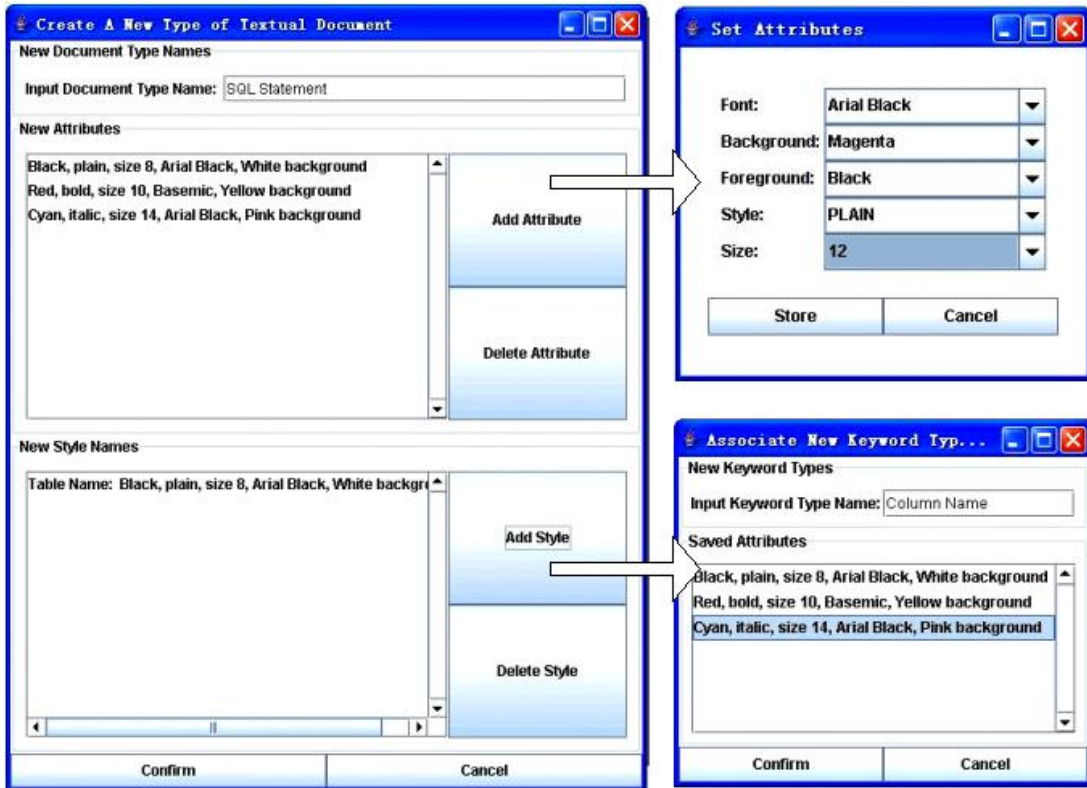


Figure 3. Creating a Document Type

Having created the document types, it is then possible to create documents. There are several ways of doing this. The simplest interface allows the user to select a fragment type and then specify the fragment text and/or position. Figure 4 shows a window in which two documents, one textual and one graphical have been created and placed in separate regions.

The right hand, graphical document in Figure 4 has been created by hand using the interface described above. The left hand, textual document, on the other hand, has been created by a program which transforms the ER diagram into a set of create table statements.

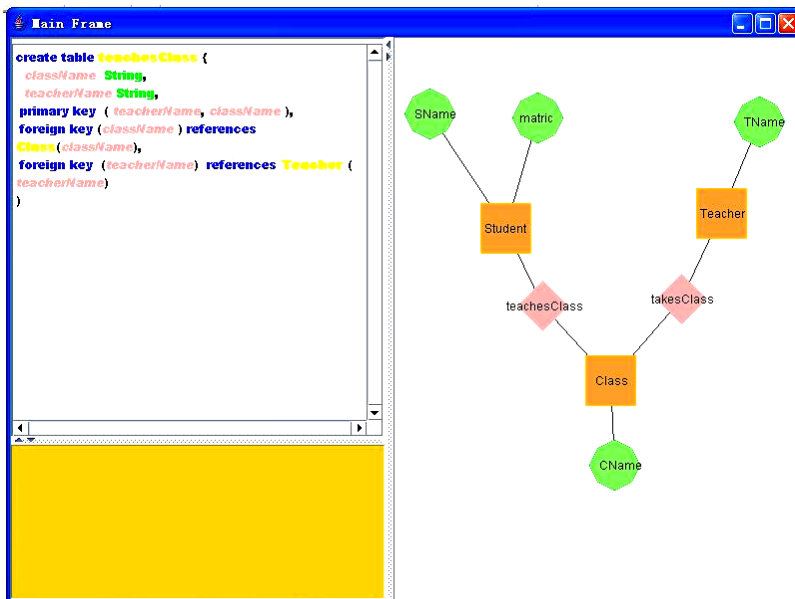


Figure 4. An Application with Related Textual and Graphical Documents

## 6. CONCLUSIONS

We have discussed the design of a framework for building the components of an immersive teaching environment which can allow the student to test out software engineering techniques and learn the concepts involved. By providing interfaces to a variety of software engineering techniques and interlinking them, the students will be able not only to explore each technique but also their inter-relationships.

The framework makes it faster and cheaper to construct such teaching applications since it includes a variety of general purpose tools for designing a screen layout and a set of document types, and then building interfaces for displaying and editing documents of those types and allocating the interfaces to regions of the screen. Since the descriptions of the interfaces, document types and underlying data are all held in a single repository, it is possible to inter-relate not just data in one interface but also indicate correspondences between the information displays of different interfaces. The example of transforming an ER diagram into a set of create table commands was used as an example.

In fact, of course, our prototype is a very crude first version. Creating a document by choosing and instantiating one fragment type after another is clumsy. We are now working on mechanisms for installing syntax directed editors both textual and graphical. We are also extending the framework to support the use of hyperlinking, glossaries, tutorials and so on, and facilities for describing processes and controlling their execution.

## 7. REFERENCES

- [1] R.L. Cooper, Q.I Cutts, C. Wang, "Immersive Learning Environments for Teaching Software Engineering", 3rd Workshop on Teaching, Learning and Assessment of Databases, July 2005.
- [2] D. Laurillard, "Rethinking University Teaching", Routledge, 1993.
- [3] C.C. Bonwell and J.A. Eison, "Active Learning: Creating Excitement in the Classroom", Washington DC, ASHE-ERIC Higher Education Report No. 1.
- [4] S. Papert, "Computers and Learning", in M.L. Dertouzos and J. Moses (eds), "The Computer Age: A Twenty Year Review", Cambridge Mass, MIT Press, 1979.