Imparting Software Engineering Design Skills

Charles Thevathayan
RMIT University
GPO Box 2476, Victoria 3001, Australia
613 999259604
charles.thevathayan@rmit.edu.au

Margaret Hamilton
RMIT University
GPO Box 2476, Victoria 3001, Australia
613 999252939
margaret.hamilton@rmit.edu.au

ABSTRACT

Teaching software engineering design to large diverse cohorts poses many challenges. Many students lacking object oriented programming skills find UML designs difficult, while large class sizes limit opportunities for interaction. The need to consider different design aspects such as reusability, extensibility etc., makes the manual assignment feedback a slow process. Moreover, such feedback is subjective, being a reflection of the individual marker's view about various design aspects. There have been little or no past attempts to provide instant and consistent feedback using constructivist tools as with programming tasks mainly because a good or correct design is difficult to define and verify.

This paper presents the result of our action research to improve student design skills. Our approach combines project-based learning with weekly quizzes, tests and active learning tasks. Quiz questions tagged with underlying concepts and cognitive levels allowed us to identify common misconceptions. Quizzes set at analysis and synthesis levels appear to foster better software design skills. Active learning tools devised helped to correct common misconceptions by providing immediate and holistic feedback. The new teaching approach helped us to improve student retention, satisfaction and performance substantially.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Design Tools and Techniques; *Object-oriented design methods; Methodologies; Representation* and I.2 [**Programming Languages and Software**]: *Expert system tools and techniques* and I.2.6 [**Learning**] *Concept learning*.

General Terms

Software Engineering Education, Active Learning, Learning Analytics, Action Research, Documentation, Program Design, Human Factors, Standardization, Programming Languages.

Keywords

UML design; Intelligent Tutoring Systems; Computer Science Education; Learning and Assessment Methodologies.

ACE '17, January 31-February 03, 2017, Geelong, VIC, Australia
© 2017 ACM. ISBN 978-1-4503-4823-2/17/01…$15.00
DOI: http://dx.doi.org/10.1145/3013499.3013511

1. INTRODUCTION

Software Engineering Fundamentals (SEF) is the only software design course core to all students doing Information Technology, Computer-Science and Software-Engineering degrees at RMIT University. It is therefore expected to nurture many critical degree level capabilities needed in software industry in addition to meeting course level objectives. The degree-level capabilities include team-working and communication skills through common frameworks, patterns and tools. For example, students are exposed to techniques for version control, and common patterns for loose coupling such as model view controller (MVC). Imparting such fundamental constructs is vital if we are to equip our students to "ride the waves" as new technologies are introduced [7].

SEF, a core to all students in the school has over 200 enrollments each semester. In the past, SEF lectures were poorly attended being viewed as a dry theory course covering many abstract software engineering concepts. Student satisfaction also was low as students with limited programming background were unable to see the relevance of many of the topics taught, including UML design, patterns and principles. Though the course covered requirements, analysis, design, processes and testing, leaving out implementation made the project experience incomplete. Large class sizes also made meaningful one-on-one interaction difficult. Similar outcomes were noted when traditional lecture-based teaching approaches were used in large design-based courses [12].

Combining design tasks with coding activities were shown to promote understanding of more abstract concepts even when a "Design First" approach was used [8]. Moreover, constantly shifting software development paradigms and technologies have made traditional lecture-tutorial based teaching practice ineffective for solving real world problems [3]. It became increasingly evident that our software engineering course must be fully restructured to allow a project driven approach, where the lecture and tutorial materials provide the necessary concepts, skills and tools to incrementally build a software system. Such an approach also allows an experiential or constructivist approach by providing students with regular feedback on their progress [3].

Our revamped version of software engineering course made the lectures more interactive by incorporating weekly diagnostic quizzes and tests. These activities allowed students to identify their own weaknesses, while the lecturer was able to identify common misconceptions. Students were also allowed to choose from a list of projects from different domains offering different levels of challenge. Students were encouraged to choose the project considering the background and interests of team members. Allowing assignment choices was shown to increase student satisfaction and learning outcomes as student interest leads to their deeper cognitive involvement [13].

Tools and techniques needed for software engineering projects such as version control, collaboration, MVC and testing were introduced through guided lab exercises. Students were also asked to follow an agile process as most students benefit from an incremental and iterative approach. Student progress is tracked through weekly contributions to collaboration tools and by monitoring the progress in the labs. Students were required to make two presentations, one midway through the project and the other at the end of the project. Our revamped course used a form of blended learning, known to be effective for large classes [12], by aligning concepts covered in lectures, weekly quizzes and tests with project milestones. The topics taught and tested therefore included design principles, patterns, processes, testing strategies as well as common frameworks for test driven development, collaboration and version control.

In this paper we report on our research undertaken during last three runs of the software engineering course using the action research methodology. Our research questions are:

- How can staff and students identify common design misconceptions in a timely manner?
- How can we help students overcome such misconceptions?

The rest of this paper is organized as follows. Section 2 highlights some of the related work in promoting a constructivist-learning environment. Section 3 presents our action research approach. Sections 4 and 5 present the details of our passive and active learning activities. Section 6 gives our assessment structure. Section 7 discusses and reflects on our experience, which is followed by acknowledgement in section 8. Section 9 concludes the presentation.

2. RELATED WORK

An intelligent tutoring system (ITS) is a computer system that aims to provide immediate and customized instruction or feedback to learners. The system allows the needs and background of the learner to be considered when creating learning tasks [15]. It requires little intervention from a human instructor once the courseware and the underlying rules are created and stored. Intelligent tutoring allows the classroom environment to be simulated by giving examples, exercises, hints and corrections. Common approaches for intelligent tutoring include both case-based and rule-based techniques [11]. A rule-based system uses a set of production rules to identify the appropriate course of action. Intelligent tutoring systems have evolved from basic answering systems dating from 1924, to present day artificial intelligence agent-based systems, which can measure various personal features of the student such as their eye-gaze [10].

In software engineering higher levels of cognitive skills are necessary as each problem presents a unique set of constraints and requirements. Software engineers must also come up with solutions, which are secure, extensible, maintainable and reusable. Moreover with the rapid changes in underlying technologies students who have a good grasp of the underlying concepts are more likely to adapt to changes. In general, students start with lower levels of activities involving comprehension and application before moving to analysis and synthesis levels. In the past, rule-based systems have incorporated levels of Bloom's taxonomy to help in classifying the difficulty level of assessment questions [8,16]. One limitation with past systems is that classification is normally limited to a single dimension. In software design, complexity increases when two or more concepts are involved. For example, a design combining inheritance and composition tends to be more complex than when they are used in isolation. In general, students should be presented activities with single concepts before composite ones. Classifications along multiple dimensions allows personalized learning paths to be created allowing average students to start with simple tasks involving a single concept at knowledge or comprehension level and progress gradually, while an advanced student may directly commence with more complex tasks involving multiple concepts at analysis or synthesis levels.

While passive learning tools help identify misconceptions, active learning tools are needed to correct misconceptions from prior experiences [2]. The use of tools which extract the semantics directly from the model allows marking to be automated [1]. Such tools allow students to learn through interactive exploring rather than by purely passive learning [6]. Moreover, such tools can help cater for diverse student cohorts by allowing students to progress at their own pace. Similar tools have been developed and used successfully in teaching programming courses in our institution [14]. However, marking design skills are more complex than programming as there is no single correct answer. Human feedback from different teaching staff often differs significantly reflecting their own biases and backgrounds leading to student frustration. While automated marking can use a common marking criteria to avoid subjective elements, developing such tools are difficult as they must consider all possible solutions. It is therefore necessary to limit the design choices when interactive tasks are designed. Moreover, catering to diverse student groups need activities that gradually increase in complexity.

In summary there is a need to extend traditional rule based systems along multiple dimensions and devise interactive design-feedback tools using an incremental and constructivist approach.

3. METHODOLOGY

The action research methodology we used allowed our teaching methods to improve over three semesters by triangulating the data obtained through performance in class tests and exams as well as response to questionnaires. Carr and Kemmis described action researchers as those who see the development of theory or understanding as a by-product of attempts to improve a situation [4]. Each action research cycle is organized into plan, act, observe and reflect stages and allows improvements to be made in each subsequent cycle based on observation and reflection. Such an iterative approach helps to improve not only the teaching methods, but also our understanding of teaching methods and the teaching environment itself [5].

Firstly we explain the operation of our software engineering fundamentals course, which runs over a twelve week semester and operates in a blended delivery format. We have two stages, each repeated ten times where students are required to attend all activities as explained below.

First Week or Stage 1 (Delivery):

1. Attend the Lecture (whole class cohort) where the overall course topics are outlined
2. Attend the Tutorials with smaller class sizes (approx. 25)
3. Take part in web-based active and passive learning tasks (individually)

Second Week or Stage 2 (Assessment):

4. A quick revision of the previous week's lecture followed by either an online test or a design exercise drawing a UML diagram for a particular scenario.

There are altogether 10 iterations of this Lecture-Tutorial-Quiz-Revision-Test cycle format for the running of the course over each semester. Students are encouraged to use the intelligent tutoring system for activities during the first week, and then in the second week the revision test covers concepts where they have demonstrated difficulties. After these discussions they are asked to either undertake an online test or a hand written test. There were no tests in the first and the final week 12.

Our action research has been undertaken over three separate semester cycles. Each action research cycle involves planning, undertaking the intervention, gathering the results, which includes the revision tests as well as separate feedback from students, and reflecting upon the impact of the intervention. In our case the intervention has been the introduction of the new intelligent tutoring system, which we have modified successively over the three cycles in response to the student feedback gathered.

The main problem that we faced at the beginning was how we could identify the root causes for student difficulties in design tasks, and how we can track both individual and group progress automatically. In the initial stage the questions were designed to identify concepts and cognitive levels where students were having difficulties. These questions permitted a passive style of learning, as the opportunities for direct interaction were limited. The data collected from these activities included responses to questions, student feedback on quizzes and explanations as well as responses collected through questionnaires.

The second part or stage 2 of the learning activities was designed to fill the knowledge and skill gap identified through the passive learning activities. The data collected for this section includes the performances in pre- and post-tests accompanying the active design activities, the student responses to questionnaires and the log of the actual student actions. The pre- and post-tests are designed to test the higher levels of learning related to the concepts. The logged activities are designed to allow us to apply learning analytics to identify common learning difficulties, which can be addressed in the following iterations.

Stage 1: Data for Individual Passive Learning

The cognitive levels tested in the first weeks were mainly through passive learning and covered the Bloom's Taxonomy levels of Knowledge, Comprehension, Application and Analysis. In the Knowledge and Comprehension levels, the students are expected to recollect and explain the concepts. These are covered mainly using multiple-choice questions. At the Application level students are expected to apply the concepts and these are tested mainly through fill in the blank type questions. The Analysis level requires students to identify the root cause of problems and these are tested through multiple choice and multiple selection based questions. The Synthesis level questions are tested through Parson's puzzle type of questions and partial completion of lines of code. Every question in the pool is tagged with the underlying concept or the group of concepts it is designed to test, the cognitive level and difficulty indices. The first index is set based on standard guidelines for Bloom's taxonomy [9,16], while the second index is set based on the percentage of students getting the correct answer.

Stage 2: Data for Active Learning through Drawing Diagrams

Tasks for active learning include designing class and interaction diagrams with or without assistance from the system. The constraints on design are either presented by pointing out the characteristics required for the activity or by supplying other diagrams, which limit the design choices available. The active learning mode gives students feedback on design decisions both at granular and holistic levels. The data collected from this section allows us to evaluate the effectiveness of the underlying design activity and the feedback mechanism.

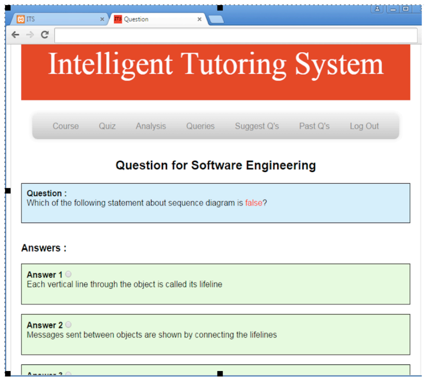
The intervention during each semester is to encourage the students to use the automated tutoring system we have developed, to introduce additional questions where students are experiencing difficulties and to help students track their own progress. Students are also encouraged to review a question and describe it as being beneficial, confusing, poorly worded or trivial. Students are invited to raise any queries on existing questions. The main activities undertaken during each semester cycle are the clarifying of the questions and the sample answers. When questions had to be changed the system allows existing responses to be retained for analysis and tracking. The system also provides the lecturer with overall summaries for each of the various topics and concepts thus allowing for timely responses.

4. DESIGN OF PASSIVE LEARNING

This course is the only software engineering course that is core to all the students in the school. A large percentage of our students hope to secure employment in software engineering related fields. If our students are to meet employer expectations it is necessary to develop higher learning levels necessary to analyze and synthesize software systems. Moreover, familiarizing students with real-world problems require devising authentic activities involving multiple concepts where some form of tradeoffs is inherent. Pedagogy however suggests students should be exposed to lower levels of cognitive activities involving only a single concept before extending to composite ones. The learning activities also need to cater for broad range of students; more advanced students need not attempt all the questions in the pool. Our intelligent tutoring system therefore caters for such varying needs by allowing different learning modes. The remaining parts of this section present the reasoning behind its design.

Sample Question at Knowledge Level

Questions at Bloom's Knowledge level are designed to test whether students can recall the definitions of basic constructs and concepts. The example below is set at the Knowledge level.



Question for Software Engineering

Question :
Which of the following statement about sequence diagram is false?

Answers :

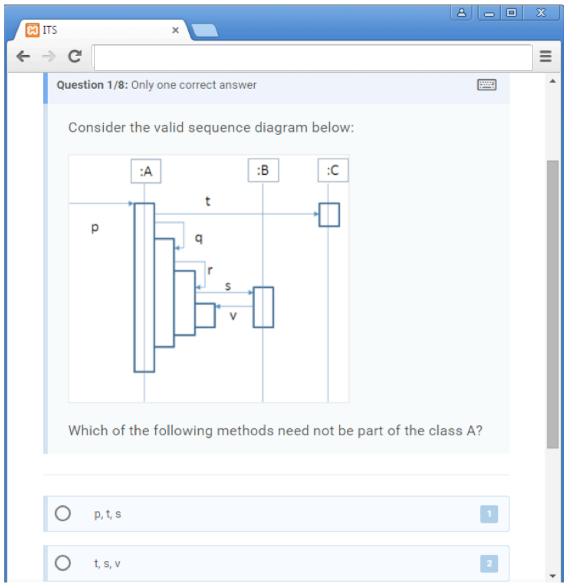
Answer 1
Each vertical line through the object is called its lifeline

Answer 2
Messages sent between objects are shown by connecting the lifelines

Figure 1: Knowledge-level Question.

Sample Questions at Comprehension Level

At the Comprehension level questions are designed to test whether students have understood the meaning of constructs and principles. Attempting these questions can familiarize students with specific terminology before they attempt more demanding questions. The Figure below shows such a question.



Question 1/8: Only one correct answer

Consider the valid sequence diagram below:

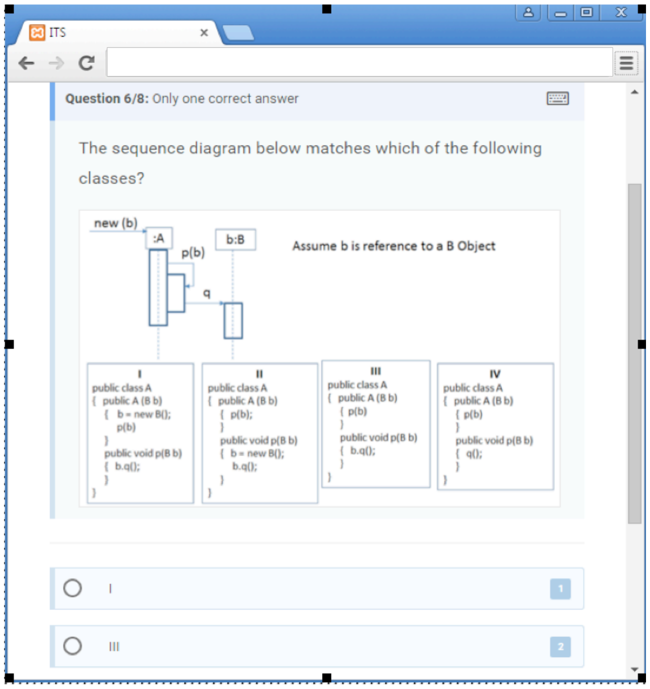
Which of the following methods need not be part of the class A?

- p, t, s
- t, s, v

Figure 2: Comprehension-level Question.

Sample Questions at Analysis Level

At the analysis and synthesis level students are tested whether they have understood the semantics of the diagram. Students are also tested to see whether they can make the connection between the code and the diagram expressing the design.



Question 6/8: Only one correct answer

The sequence diagram below matches which of the following classes?

Assume b is reference to a B Object

I
```
public class A
{  public A (B b)
   {  b = new B();
      p(b)
   }
   public void p(B b)
   {  b.q();
   }
}
```

II
```
public class A
{  public A (B b)
   {  p(b);
   }
   public void p(B b)
   {  b = new B();
      b.q();
   }
}
```

III
```
public class A
{  public A (B b)
   {  p(b)
   }
   public void p(B b)
   {  b.q();
   }
}
```

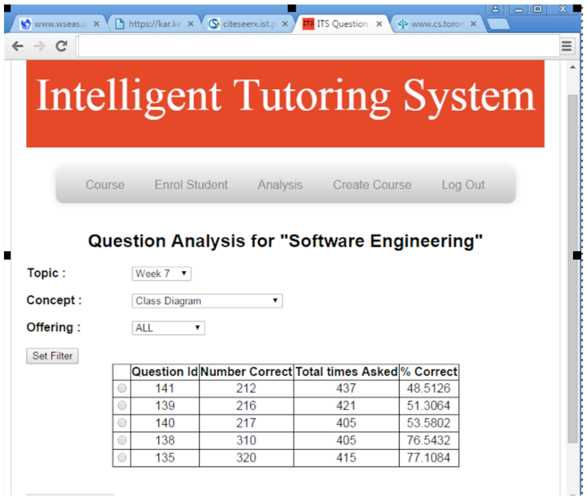
IV
```
public class A
{  public A (B b)
   {  p(b)
   }
   public void p(B b)
   {  q();
   }
}
```

- I
- III

Figure 3: Analysis-level Question.

Concepts and Topics

Every question must be tagged with a topic and at least one concept it exemplifies. A question in a given topic can also cover multiple concepts, however all the concepts must be covered in earlier topics to prevent a student being presented a concept that is not yet covered in the lectures.



Intelligent Tutoring System

Question Analysis for "Software Engineering"

Topic : Week 7

Concept : Class Diagram

Offering : ALL

	Question Id	Number Correct	Total times Asked	% Correct
○	141	212	437	48.5126
○	139	216	421	51.3064
○	140	217	405	53.5802
○	138	310	405	76.5432
○	135	320	415	77.1084

Figure 4: Introductory Screen for the Intelligent Tutoring System.

Conceptual Dependencies and Learning Modes

A concept can be dependent on other concepts already covered. Capturing conceptual dependencies allows questions to be generated in a specific order in the chosen learning mode. If an advanced student chose a top-down mode and gets a particular question wrong, the system automatically generates questions covering dependent concepts, until the root cause for the learning difficulty is identified. In the progressive mode students are presented with less demanding tasks first. Moreover, questions involving base concepts are presented before composite ones.

Difficulty and Discrimination Indices and Range Mode

The difficulty index has two parts; the first one reflects the level of difficulty perceived by the creator, the second one reflects the percentage of students getting it correct. The discrimination index reflects how well the question discriminates between good and average passing students. These indices can be used together to create balanced quizzes if and when these data are available. Moreover students can specify the difficulty and discrimination range if they choose a particular Ranged Mode. This option became available only in the second iteration as these indices are derived based on previous iterations.

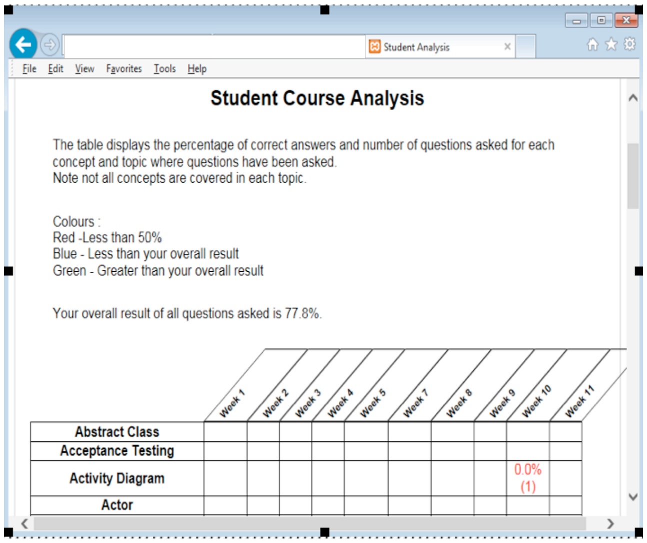
Two-way Interaction

Each time a student submits a response, they receive an immediate reply about whether the answer is correct. Students are also able to see the correct answer and the worked solution. The system is designed to promote two-way interaction. Any student who fails to understand the solution may forward a query for the teaching staff. Students are also encouraged to rate the questions as well, allowing poorly worded or confusing questions to be replaced.

Staff and Student Analysis

Staff can view how students are performing during a particular semester's offering, in both topic and concept levels. This allows staff to correct any misconceptions in the lecture making the learning style a blended learning mode. The system also allows staff to set a short quiz to be taken during the lecture (through mobile devices) to ascertain the levels of student understanding.

Students are able to view their progress at any time. The Analysis option shows how they are performing in each individual concept and topic. Staff can assist a student better if they know the main areas of particular student difficulties and the system can provide this information, see Figure 5.



Student Course Analysis

The table displays the percentage of correct answers and number of questions asked for each concept and topic where questions have been asked.
Note not all concepts are covered in each topic.

Colours :
Red -Less than 50%
Blue - Less than your overall result
Green - Greater than your overall result

Your overall result of all questions asked is 77.8%.

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 7	Week 8	Week 9	Week 10	Week 11
Abstract Class										
Acceptance Testing										
Activity Diagram									0.0% (1)	
Actor										

Figure 5: Student screen showing their individual progress through the concepts and modules.

At the end of each semester we conduct an anonymous web survey on the use of ITS system with the following questions:

1. ITS required me to review my course materials regularly
2. ITS helped me to do well in the weekly tests
3. I spent on average 15 minutes or more on ITS each week (from week 4)
4. The option to view questions by topics was useful
5. The option to raise queries and get feedback was useful
6. The option to analyze performance across topics and concept was useful
7. The option to view performance in past questions was useful
8. ITS had questions with different levels of difficulty
9. ITS system was easy to use
10. Different modes (Progress, Top Down) allowing different difficulty levels was useful
11. The ITS system was responsive enough
12. ITS helped me to improve my understanding
13. I was able to access ITS from mobile devices satisfactorily
14. ITS should also be used in other courses where there are difficult concepts
15. ITS can be useful for revision and for identifying areas of weaknesses.

Students were asked to rate the questions above using a Likert scale where 1 was Strongly Disagree and 5 was Strongly Agree. We present their responses in Table 1 below.

Table 1: Student feedback after Cycle 1 after using the Intelligent Tutoring System, where 1-Strongly Disagree and 5-Strongly Agree

Statement\ Likert Rank	**1**	**2**	**3**	**4**	**5**
1	0	4	8	16	24
2	1	0	5	16	30
3	0	0	15	13	24
4	1	4	9	17	21
5	1	3	12	14	22
6	1	2	11	19	19
7	0	3	9	14	26
8	1	3	10	15	23
9	0	5	9	14	24
10	7	5	12	15	13
11	0	4	16	12	20
12	0	1	13	14	34
13	3	4	23	13	9
14	1	2	7	15	27
15	0	3	6	17	26

The results for the ITS system showed most students liked the tool though the user interface could still be improved. From the ITS survey it is evident the feedback from the ITS system helped them with the weekly test in the following week. Over 80% of the students wanted the system to be extended to other courses and found ITS to be particularly effective for revision and for identifying their areas of weaknesses.

5. DESIGN OF ACTIVE LEARNING

Though passive learning tasks may provide staff and students with an effective way to understand learning difficulties, they are of limited use for correcting an invalid mental model or fostering a valid mental model. Analysis of past ITS data clearly showed students struggled with object-oriented design. A number of factors can be attributed to poor performance in object-oriented design. Students are unable to get any early feedback unlike programming tasks where students can easily test their programs using predefined test cases. Secondly being a core course for all students, students from specific pathways may not have adequate knowledge of object-oriented programming. Thirdly students used to having a single correct answer find it unsettling to be told there can be multiple valid answers providing different tradeoffs. The challenges facing the instructor are to

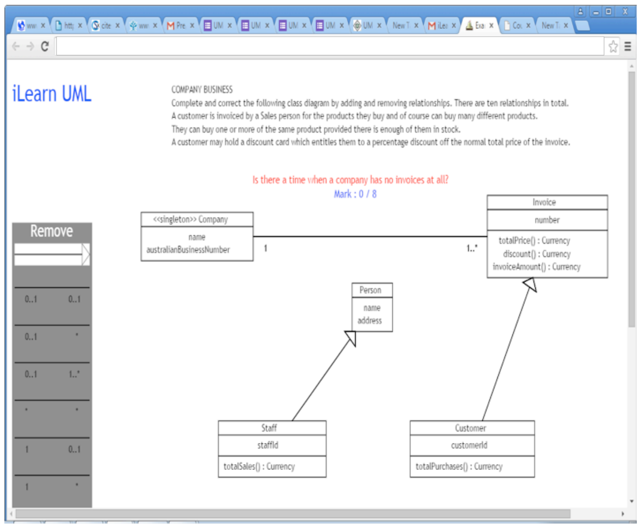
- Provide immediate feedback on student designs
- Allow multiple correct answers but provide holistic feedback from different perspectives.
- Come up with increasingly open-ended design questions

There were two key active learning tasks, involving class and sequence diagrams. These were assessed in separate cycles of the action research and we discuss them in the next two sections.

Tool for the Active Task Involving Class Diagrams

The activity for the class diagram in Figure 6 is specified below:

"Complete and correct the following class diagram by adding and removing relationships. There are ten relationships in total. A customer is invoiced by a sales person for the products they buy (there can be one or more product). They can buy one or more of the same product provided there are enough of them in stock. A customer may hold a discount card which entitles them to a discount off the normal total price for the invoice."


COMPANY BUSINESS
Complete and correct the following class diagram by adding and removing relationships. There are ten relationships in total.
A customer is invoiced by a Sales person for the products they buy and of course can buy many different products.
They can buy one or more of the same product provided there is enough of them in stock.
A customer may hold a discount card which entitles them to a percentage discount off the normal total price of the invoice.

Is there a time when a company has no invoices at all?
Mark : 0 / 8

Figure 6: Designing UML Diagram Screen.

The first task was designed to teach class diagrams. Many students experience problems identifying associations and the multiplicity based on the scenario specified. The class diagram tool is designed address these concerns by presenting increasingly complex class diagrams. Students are provided a user interface to:

- Add / remove generalizations
- Add / remove associations
- Specify multiplicities.

The feedback was provided for each invalid and missing association and generalization relationship and multiplicity. Students were allowed to start with simple class diagrams before proceeding to more complex ones. The user interface allowed students to use mouse and button clicks as shown in the Figure 7. Students were allowed to use the tool during the lab sessions and from their homes and mobile devices. Students were asked to complete a survey at the end of the activity. The survey was conducted at the end of the second action research cycle and was completed by 92 students.

The responses for the first four questions above were options on a Likert scale where 1 was Strongly Disagree and 5 was Strongly Agree. We present the answers in Table 2 below.

Table 2: Student feedback after Cycle 2 after using the Intelligent Tutoring System, where 1-Strongly Disagree and 5-Strongly Agree

Statement \ Likert Rank	**1**	**2**	**3**	**4**	**5**
The tool was easy to use	5	32	23	28	4
The tool can clarify associations	1	3	16	46	26
The tool can clarify multiplicities	0	2	10	42	38

Table 3: Written Feedback From the Survey after Cycle 2

Good helpful concept, lot of potential but feels unresponsive and frustrating to use in its current state
A legend on the side of the page listing instructions
Having hints that you can turn on and off is helpful
More feedback is needed on incorrect associations
It was helpful but I wanted more visual feedback after each step
Easier GUI please

From Cycle 2, the main feedback on the tool was the need to improve the user interface. Over 80% of the students said the tool could help them learn associations and multiplicities. Students also liked the way their activity was marked. The next phase of the tool development therefore focused on improving the user interface adding extra panels, legends and hints. We also added another mode where instant feedback could be provided.

Tool for the Active Tasks involving Sequence Diagrams

In Cycle 3, we modified the intelligent tutoring system to include active sequence diagram tasks. Past data based on assignments and exams reveals the sequence diagram as one of the most difficult UML diagrams for our students. The difficulties are not only in using the notation itself but are also the result of failing to comprehend how objects interact by passing messages. These mistakes become more pronounced when students are asked to design class and sequence diagrams for a common scenario.

In general many of our students are weak in understanding distributed design which reduces the coupling between classes. Students with limited object oriented programming background can find it difficult to make the connections when methods return references to other objects. The design focuses on creating increasingly complex scenarios. Initially class diagrams allow students to use a centralized design by providing the necessary methods while subsequent scenarios require students to use either a centralized or distributed design. The holistic feedback uses common metrics to identify which designs are better from a maintainability perspective.

Improvements for the New Tool

There were a number of improvements incorporated into our modified Sequence tool and based on the feedback collected from the Class Diagram tool.

- For each model (class diagram) there are a number of use cases to be converted into sequence diagrams.
- Feedback is given after adding each message.
- Invalid messages are automatically rejected based on the model specified by the class diagram.
- Students can switch between the model and the sequence diagram they are required to edit.
- Holistic feedback on the quality of design is given on completion of the task.
- Both pre- and post-tests are given to assess whether there is any significant improvement in student performance.

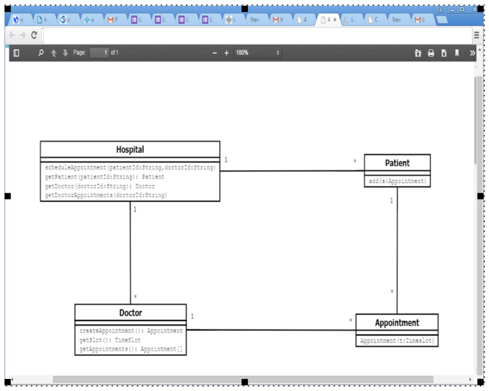

Page: 1 of 1
180%
Hospital
scheduleAppointment(patientId:String,doctorId:String)
getPatient(patientId:String): Patient
getDoctor(doctorId:String): Doctor
getDoctorAppointments(doctorId:String)
1
*
Patient
add(a:Appointment)
1
*
1
*
Doctor
createAppointment(): Appointment
getSlot(): TimeSlot
getAppointments(): Appointment[]
1
*
Appointment
Appointment(t:Timeslot)

Figure 7: The Class Diagram Constraining the Sequence Diagram

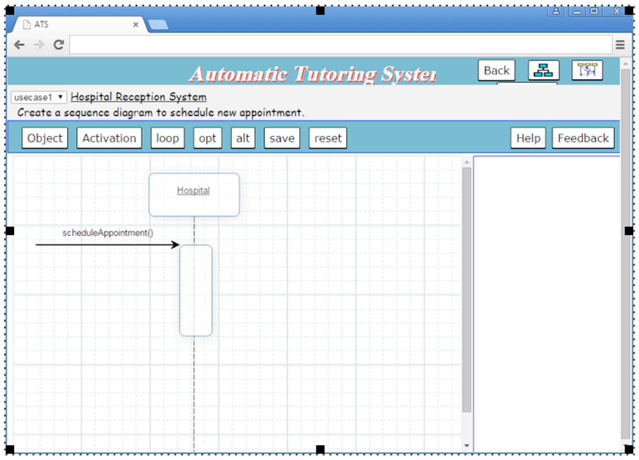

ATS
Automatic Tutoring System
Back
usecase1
Hospital Reception System
Create a sequence diagram to schedule new appointment.
Object
Activation
loop
opt
alt
save
reset
Help
Feedback
Hospital
scheduleAppointment()

Figure 8: Designing the Sequence Diagram

At the end of each semester we ask the students various questions about the ITS and their understanding of the software engineering concepts. After Cycle 3, we asked the following questions:

1. I find learning some of the UML diagrams difficult
2. I find the relationships between different UML diagrams difficult to understand.
3. I need tools that can give feedback on UML designs.
4. I would like similar activities for other UML diagrams

The responses on a Likert scale where 1 was Strongly Disagree and 5 was Strongly Agree are presented in Table 4.

Table 4: Student feedback after Cycle 3 after using ITS, where 1-Strongly Disagree and 5-Strongly Agree

Statement\ Likert Rank	1	2	3	4	5
1	2	2	13	16	11
2	2	8	14	15	5
3	4	0	5	13	22
4	3	1	10	14	16

In the case of sequence diagrams we wanted to find out whether attempting the active tasks impacts the student performance significantly, by comparing the pretest with the post-test results. Eight pre- and post-test questions were chosen in random order from a pool of 16 questions at the levels of Comprehension, Application and Analysis. Each test contained the same distribution across the cognitive levels. Out of the 243 students in the course 94 students volunteered to undertake the pretest. From this group only those 68 students who proceeded to do the activity and the post-test were used in the analysis.

Figure 9 shows there were a major shift in the overall performance across the whole range of student abilities. The number of students in the 90-100% range went up from 0 to 6 after the active tasks while the number in the less than 50% mark range decreased from over 30 to less than 15. Based on these values we reject the null hypothesis "No significant improvement after the active task" as the Chi-square P-value of 0.000269 is less than 0.01. Students were also able to perform well on the written class test on sequence diagrams with average marks at 71%.

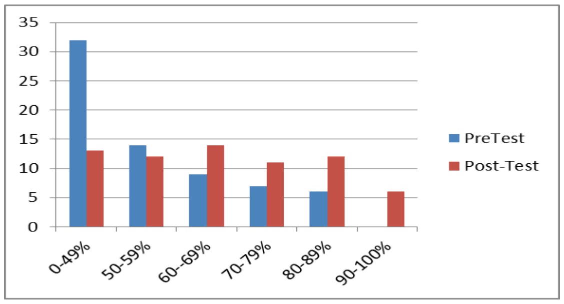


Figure 9: Pre and Post Test Results from the Sequence Diagram Activities from Cycle 3

6. ASSESSMENTS

The software engineering fundamentals course assessment is comprised of 3 parts. The class tests and team project accounts for 30% each, while the exam a hurdle, forms the remaining 40% of the overall marks. The exam was made up of 3 parts: multiple choice questions; short answer questions; a major design exercise covering most of the UML diagrams. One reason to introduce class tests was to improve student attendance in the lectures. Such tests also motivate students to do regular work and to take remedial steps early in the semester. Six of these class tests were delivered through another web-based system using wireless and mobile devices. The number of questions in these tests varied from 10 to 20 and the average time allocated was 15 minutes. Students were allowed to see their results immediately after submitting their responses.

Our marks for the project based learning consist of progress marks and marks for specific milestones following an agile approach. The first milestone required a working prototype and a number of test cases. The second prototype required the working product as well as all the UML diagrams. Marks were awarded for both the team and the individual parts. Marks were also awarded each week in the lab for showing progress in terms of design, implementation and testing. The progress was monitored through collaboration and version control software as well as face-to-face by the tutors. Progress marks played an important role in ensuring regular attendance and faster resolution of any team problems.

7. DISCUSSION AND FUTURE WORK

Software engineering fundamentals is a core course for IT, Computer Science and Software Engineering students. We cater for such a diverse group by using a multimodal approach which combines project milestones, progress reports, use of collaboration tools, lecture materials outlining main principles, design exercises in the tutorials, quizzes with different learning modes, active design learning tools and regular class tests.

The intelligent tutoring system (ITS) was designed to provide personalized learning pathways by tracking the progress of each individual student. The ITS courseware consists of 250 questions divided into various cognitive levels, concepts and difficulty levels. Performance in analysis and synthesis level ITS questions correlated most closely with open-ended design questions in the exam. On average over 80% of the student cohort attempted the quizzes each week, partly because similar materials were assessed in subsequent week through web-based tests or a paper based design tests. Each web-based test question covers one or more concepts. Written-tests focus mainly on UML design diagrams. These tests and associated quiz related activities have helped to improve the attendance rate, class interaction and the performance in the course as a whole with average attendance rate over 80%. Students were allowed to actively participate with the ITS system by raising queries about difficult or confusing questions or explanations. Responding to these queries often resulted in improving the way the questions and explanations are phrased. ITS also provides teaching staff valuable timely feedback on student progress. Students too are able to track their progress effectively. Year-end survey shows students found the ITS system very helpful for learning concepts, revision and project activities.

Despite many benefits, ITS also presents many challenges; quizzes relating to coding and testing must be updated as technologies, notations and languages evolve; its educational benefits greatly depends on the quality of courseware. Crafting questions to demonstrate specific concepts or patterns is a laborious task. One benefit of such a cloud-based system is that courseware developed can be shared with any other institution across the globe. Moreover, staff from different universities can collaborate in educational research by sharing resources and by analyzing student-learning outcomes. Any academic staff can request access to the system by contacting one of the authors.

Incremental visual constructivist tools have helped to improve student engagement and performance in programming courses. Creating such a tool for UML design diagrams is difficult, as the tool must represent the semantics of the underlying model. The active learning tools we devised therefore restrict the design choices. These restrictions can be gradually relaxed as students become more confident. Holistic feedback relied on our predefined metrics. Currently we are working on improving the initial versions of these tools.

8. ACKNOWLEDGEMENTS

We want to acknowledge and thank Phillips Reynolds who was the main developer of our ITS system. We also want to thank Naveen Bajaj, Keith Foster and Quan Do.

9. CONCLUSION

Large classes make it difficult to give adequate manual feedback in the early formative stages. In the past, much work has been done to provide immediate feedback on programming tasks. Automated feedback is even more important when students begin to create design diagrams, which are generally more abstract. We have developed a multimodal approach to foster design skills relevant to industry by combining a project driven approach with automated feedback mechanisms and quizzes. Weekly project progress demos allowed students to learn from other team members and through tutor feedback. The quizzes designed to highlight common design errors, allowed students to improve their design skills. Classifying quiz questions into concepts and cognitive levels allowed teaching staff and students to identify problem areas. The active learning tasks supplementing these quizzes helped to correct any misconceptions early through immediate feedback. This approach also facilitated an incremental approach by reducing the number and types of constraints attached to design tasks. The holistic feedback took into account non-functional aspects such as performance, maintainability and extensibility thus prompting students to improve their designs. Our multimodal approach combining individual and team learning techniques helped us to improve student engagement, satisfaction and performance substantially.

10. REFERENCES

[1] Ali, N. H., Shukur, Z., and Idris, S., 2007. Summary Assessment System For UML Class Diagram Using Notations Extraction. International Journal of Computer Science and Network Security, 7(8), 181-187.

[2] Baser, M., 2006. Promoting conceptual change through active learning using open source software for physics simulations. Journal of Educational Technology, 22(3).

[3] Brodie, L., Zhou, H., and Gibbons, A., 2008. Steps in developing an advanced software engineering course using problem based learning. Engineering Education, 3(1), 1-12.

[4] Carr, W. and Kemmis, S., 1986. Becoming critical: education knowledge and action research. Lewes, UK: Falmer Press.

[5] Clear, T., 2004. Critical Enquiry in Computer Science Education. In Falmer, R. London: Taylor and Francis Group.

[6] Dym, C. L., Agogino, A. M, Eris, O., Frey, D. D., and Leifer, L. J., 2005. Engineering Design Thinking, Teaching, and Learning. Journal of Engineering Education, 94(1).

[7] Ghezzi, C. and Mandrioli, D., 2005. The challenges of Software Engineering Education. ICSE 2005, 115-127.

[8] Giordano, D. and Maiorana, F., 2014. Teaching "Design First" Interleaved with Object Oriented Programming in a Software Engineering Course. Paper presented at the Global Engineering Education Conference (EDUCON), Istanbul.

[9] Haris, S. S. and Omar, N., 2012. A rule-based approach in Bloom's Taxonomy question classification through natural language processing. Paper presented at the Computing and Convergence Technology, Seoul.

[10] Jaques, N., Conati, C., Harley, J., and Azevedo, R., 2014. Predicting Affect from Gaze Data During Interaction with an Intelligent Tutoring System. Paper presented at the Intelligent Tutoring Systems.

[11] Noha, N. M., Ahmad, A., Halim, S. A. and Ali, A. M., 2011. Intelligent Tutoring System Using Rule-Based And Case-Based: A Comparison. Paper presented at the International Conference on e-Learning, Bandung.

[12] Oliver, R., 2005. Using a blended learning approach to support problem-based learning with first year students in large undergraduate classes. Edith Cowan University.

[13] Tobias, S., 1994. Interest, Prior Knowledge, and Learning. Review of Educational Research, 64(1). Supporting Diverse Novice Programming Cohorts through Flexible and Incremental Visual Constructivist Pathways.

[14] Thevathayan, C., and Hamilton, M, 2015. Supporting Diverse Novice Programming Cohorts through Flexible and Incremental Visual Constructivist Pathways. Paper presented at the ACM Conference on Innovation and Technology in Computer Science Education, Vilnius.

[15] Tuaksubun, C. and Mungsing, S.,2007. Design of an Intelligent Tutoring System that Comprises Individual Learning and Collaborative Problem-Solving Modules. eLearning for Knowledge-Based Society, Bangkok.

[16] Whalley, J. L., Lister, R., and Thompson, E., 2006. An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using Bloom and SOLO Taxonomies. Paper presented at the ACE2006, Hobart.