Capturing Collaborative Designs to Assist the Pedagogical Process

Thomasson, Benjamin Jason Tom; Ellis, Wayne; Thomas, Lynda; Ratcliffe, Mark Bartley

Publication date:
2003

Citation for published version (APA):
Thomasson, B. J. T., Ellis, W., Thomas, L., & Ratcliffe, M. (2003). *Capturing Collaborative Designs to Assist the Pedagogical Process*. http://hdl.handle.net/2160/249

Capturing Collaborative Designs to Assist the Pedagogical Process

Mark Ratcliffe, Lynda Thomas, Wayne Ellis, Benjy Thomasson
Department of Computer Science
University of Wales, Aberystwyth, UK
{mbr, ltt, wwe8, bjt98}@aber.ac.uk

ABSTRACT

This paper describes a project being undertaken at the University of Wales, Aberystwyth that captures students' designs in an attempt to improve the pedagogy. To enhance their understanding of object oriented programming, students are given an environment that enables distance working and encourages group collaboration whilst capturing all aspects of development of their designs. To enhance the teaching of programming and software design, instructors are given access to complete design histories enabling them to better understand how the students view their design processes, the problems that arise and the steps that they take to resolve them.

Categories and Subject Descriptors

D.1.5 Object-oriented Programming.

General Terms

Design, Experimentation, Human Factors.

Keywords

Pedagogy, design capture, learning to program.

1. OVERVIEW

Many academics and industrialists are beginning to realize that the way we are educating software engineers is flawed. Ratcliffe coordinated a session on this in the USA early in 2002 [1]. A straw-poll survey suggested that in both the UK and the USA about 30% of students studying Computer Science are failing to grasp the basics.

At first, there seems to be little alternative to providing one-to-one tuition that points out where a student is going wrong at the critical time. Indeed determining the critical time is itself of course a tricky pedagogical issue. Unfortunately this level of tuition is just not practical in today's academic environment of growing student numbers and limited resources.

The Department of Computer Science at the University of Wales, Aberystwyth is developing a tool where we aim to provide the equivalent of this one-to-one tuition in the form of a software system. The system gives personalized advice while providing savings in terms of time and money. It also allows, and in fact encourages students to work in a collaborative way.

This system, referred to as VorteX (Visually Oriented Training Experience), is a fully interactive, collaborative design capture and feedback system. It captures the design process of cooperating novice designers in order to refine our knowledge of the way in which the design learning process works. In this way we are able to provide automated support that is both more focused and relevant.

The knowledge gained from having students work through a specified set of case studies is currently being used to populate a case-based system capable of assisting novice software engineers to develop higher quality designs. It is enabling us to acquire a much better understanding of the student's perception of design and the learning process that it involves.

As the system captures collaborative software designs, ultimately it should be able to simulate other group players and thus give individuals experience of designing within a team, even if one is not present.

The long-term aim is to develop a more generalized system that has an underlying model of the design learning process. At that point it is hoped that it can break free of the captured cases and be capable of advising students in the general development of their designs.

2. THE PROBLEM

Current teaching is less effective than it should be

Software development is a relatively young engineering discipline, yet there have already been many fundamental changes in the techniques employed for the basic development of software systems. Object Oriented Design is the latest paradigm and is now commonplace in most courses on software development. This technique is claimed to be one of the most natural forms of development, modeling closely the way in which systems exist in the real world, yet there are major problems in teaching the technique. At a recent conference of the *Learning and Teaching Support Network for Information and Computer Sciences* [2], there was wide consensus that the success rate of teaching Computer Science freshmen is very poor. An international review of first year students' programming skills reached a similar conclusion [3], that about 30% of students are failing to grasp the basics of design. An informal birds-of-a-feather group at the 2002

SIGCSE conference [1] raised the point that perhaps we are failing to get across the principles of Object Oriented Programming.

Many different approaches have been used to assist novice programmers in their attempts to learn the Object Oriented Design process but as demonstrated in a recent paper [4], if we examine typical advice that is given to designers and programmers by textbooks and references, we see many appeals to experience.

"The first step in actual class design is to find the primary objects" [5]
"Identify the classes and objects at a given level of abstraction"[6]
"The content of an object model is a matter of judgment …" [7]
"As analysts experienced in [design…], we recognize certain patterns" [8]
and the list goes on. This is a chicken and egg problem. How are students supposed to apply judgment in the absence of such experience?

Tutors' experience is a barrier to student learning
Whilst novice programmers are often able to recite the techniques necessary to approach the design process, when things get a little hard they will usually turn to more experienced tutors to decrypt what is required. Most tutors have years of valuable experience building their understanding of the design process and naturally base their tuition techniques upon it. Unfortunately, although the tutors' experience is the key to their own success, it is also a complicating factor. The lack of experience on the students' part can put up a significant barrier between the tutor and student. It could be the very reason why the tutor cannot appreciate the real difficulty that the students face.

The authors are not trying to claim that we need to replace the tutors – far from it; in many cases they are the right people to convey this knowledge. What we are saying is that additional assistance can be provided to help the students and we hope to prove that this assistance can emanate from the student body itself.

3. IDENTIFYING THE DIFFICULTIES

As a first step in identifying the real problems that students face in producing their solutions, we thought it useful to try to capture their decisions. The idea behind VorteX is to monitor the students as they design a small number of software systems. Each step of the way, as the users add and remove classes, add and remove methods, merge classes, change the signatures of methods, and so on, VorteX captures the modification and attempts to obtain as much explanation as is possible.

4. CAPTURING THE DESIGN

Capturing the design process is fairly straightforward. Exactly what is done with the information is much more challenging and is explained later in this paper.

The first stage involves presenting students with example specifications. There are five such specifications used in the first year software development class. The following is one such abbreviated example:

"The government of Elwha, a small island in the Pacific has commissioned you to build a software system to manage both their air traffic and baggage control systems.
The airport consists of two separate terminals separated by a distance of about 3 miles. The first terminal has two runways. The second terminal has three.
….
You are required to produce class diagrams that will support the following operations:
Assign an incoming flight to a particular runway
…..…"

This is the kind of specification that might be given to students in their first few weeks of software design. It takes a much-simplified example of the real world and is not looking for any major inspiration on the student's part. Its purpose is to provoke a simple breakdown of classes that might be provided together with identification of methods within those classes.

One might argue that no novice student is going to be willing to put up with a system that requires an explanation for every move that is made. Fortunately because these explanations feed directly into the development by generating associated in-line documentation, most students are content to provide this information. After all, providing documentation is an unpopular requirement for most novice programmers; anything to assist is held in high regard.

Capturing student input is not a new idea. Similar work is underway in the UK through the AESOP [9] project though this is largely concerned with recording lower level information such as keystrokes; it is not applying case-based or model-based technology to solve students' problems. As far as we can ascertain, there is little work being carried out in capturing collaborative work.

To ensure that the student is aware of all that is being captured, the information is shown in the log window as is depicted in the lower section of Figure 1. Even without any further case-based analysis, this capture is itself a useful design aid. Both students and their tutors are able to scroll through the log information to see the justification for previous changes and to identify the source of any misconceptions.

Despite the fact that justification is required for all changes made in the editor, the problem remains that we might still not be capturing all that is going on in the mind of the student carrying out the design. It may be that various alternatives are considered and that only when a student is satisfied will he/she commit to a possible design. What we really need is an extra level of indirection, another kind of sounding board which can itself be captured.

5. ENHANCING CAPTURE THROUGH GROUP WORK

Cooperative Design is a good thing.
If we can encourage students to cooperate we might get more information out of them. Work undertaken at the German

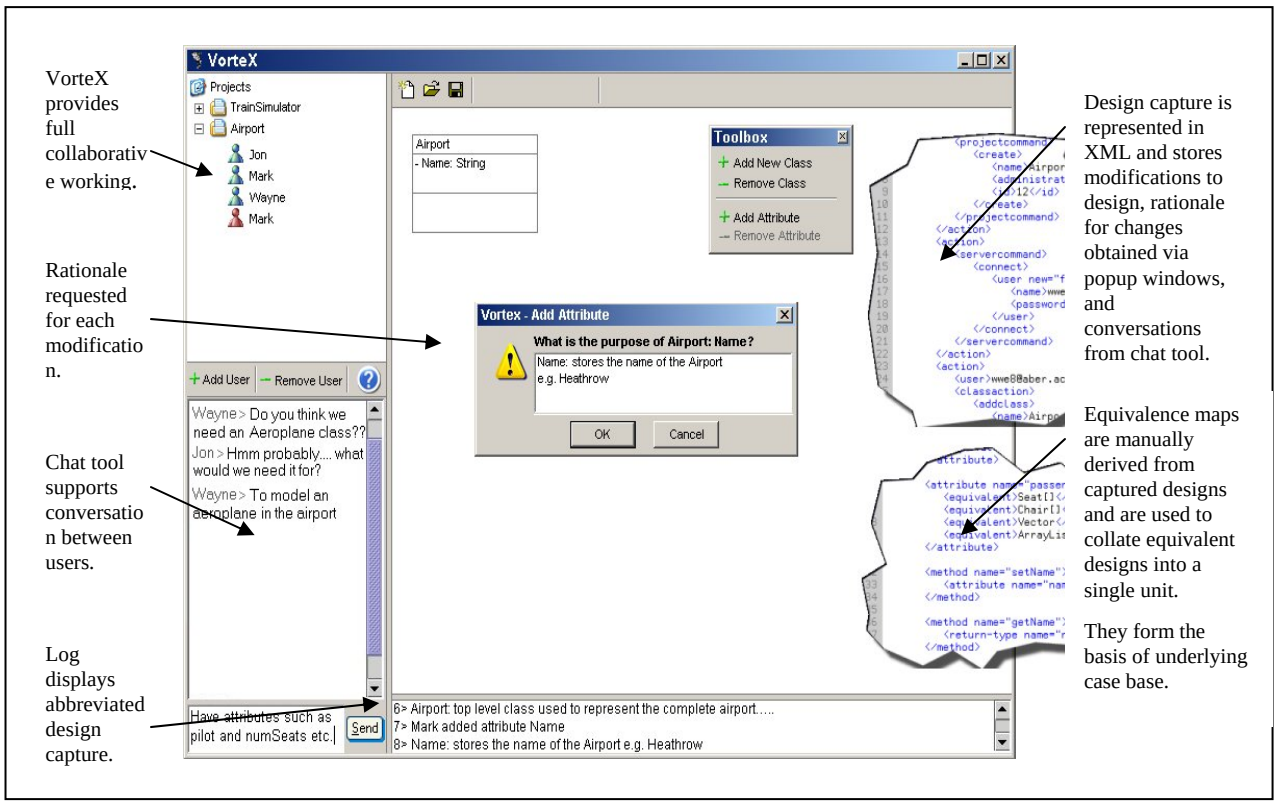

Figure 1: VorteX adding an attribute

National Research Center for IT demonstrates the benefits of cooperative learning in design [10], in particular it has the added advantage of causing students to justify their design decisions and reflect upon them. Research has shown that knowledge alone is not sufficient for successful problem solving in a domain; the student must also choose to use that knowledge, and to monitor the progress being made [11]. The learning and construction of new knowledge structures requires similar self-awareness and reflection.

VorteX supports cooperative design and as in single user mode captures the entire communication process. The first group member to start up VorteX is essentially the administrator and he/she is able to register any other users who may wish to contribute to the project. Any of these other users can be allocated administrator privileges should the originator move away from the team or be unavailable. To support communication, free text is captured in the form of a chat tool, but by its very nature is difficult to parse. To ensure that the basic design capture is retained, group user mode still requires all changes to the design to be justified through the pop up windows. Figure 1 again shows this in its centre.

The facility to produce designs cooperatively both simulates "real-world" team design contexts and also brings some of the flavor of pair programming [12] to a VorteX session. Students have responded well to pair programming at Aberystwyth [13] and although the VorteX experience allows parallel working[1], it does provide the opportunity for sharing ideas, a quality which students cite as a real advantage in pair programming.

6. WHAT VORTEX CAPTURES

VorteX uses an XML definition to record a complete path of progression through the group's design. As the design changes, VorteX records every action and logs the results as a design capture log. From a pedagogical point of view, a design capture log allows us to see how the design evolves, where and when changes occurred, why they happened and who carried out the modifications. The actual log displayed to the user is parsed into a much more user-friendly version, in a form designed to be suitable for novice programmers.

7. HOW VORTEX IS BEING USED

VorteX makes the entire history of the development available to the tutor showing how the design evolved from its initial conception. Current developments are planning to extend VorteX to provide an animation of the development using the design capture log. Using this facility it will be possible to step through a development examining what happened at each stage. One can see that this could have advantages both to the students themselves (particularly within a group environment) and to the tutor wishing to give feedback to a student.

[1] Rather than pair programming in which students work on one machine.

Of particular pedagogical interest are any planes of unconformity that exist in the development. These represent radical design changes rather than gradual evolution and usually indicate a complete change of design. Although unfortunate, it is often the case that less experienced advisors/tutors will suggest a whole new approach to a confused student rather than attempting to explain what is wrong with their current design. This approach can often leave students wondering what was wrong with their approach.

When it comes to evaluating the student work, a project developed through VorteX provides much more information to the tutor than is typically available. VorteX is server based, and maintains a centrally stored model of all the projects; in this way an instructor is able to get detailed information on an individual's progress. By selecting a particular project, the instructor is able to access the design in its current state. The development log shows the design history and any chat tool dialog that might be available shows the conversation that took place prompting the developments. Extensions to the VorteX application provide simple log assimilation facilities presenting the tutor with graphical representations of the work effort applied by each group member. Statistics are provided for individuals and complete groups. This helps to complement existing methods of project assessment through identification of the work carried out by each pupil. Through automated analysis, we can see which students performed more important actions such as the identification and addition of classes and attributes, in comparison to the group members that implemented the bodies of the classes. With this level of analysis we can aim to identify the abilities of individual group members and possibly even the role they lead in their team.

8. PRELIMINARY RESULTS

The project is still only in its infancy and full deployment of the tool is still a couple of months away, yet already we have some interesting results. The first specification deployed on a class of 115 students resulted in only 20% of students producing a design anything like that expected by the lecturer. Providing information such as this can really help a lecturer assess the progress of a class. Over the next few months as the captured designs are analyzed we expect to get more information that will challenge our perceptions of how successful courses really are. It is already clear that to be most effective, VorteX needs to be deployed to other institutions that use different methods and styles of teaching. It appears that as a course progresses, the designs produced by students get closer and closer to those of the lecturer teaching the course. Whether this means that the students are producing better designs or simply that they develop an understanding of what the lecturer wants is still up for investigation.

9. USING THE CASE HISTORIES

VorteX has so far been deployed in capture mode only – in order to populate the case-base. The five project specifications have been carefully developed and released one at a time to over 100 freshmen on the introductory programming course. These case studies have been chosen to encourage team working so as to enhance communication and make it easier to capture the feedback loop fundamental to the learning process.

Even though the case-base functionality is not yet available students have been able to benefit from VorteX, working with others and enhancing their design skills. The ability to trace back through designs is of real pedagogical value.

At the same time as deploying the specifications, we have developed equivalence maps for each project specification. These are fundamental to the case-based analysis and were initially produced by the tutor to represent possible solutions to the specification. These solution maps are expressed in XML and define the classes, their attributes and their methods. Against each identifier is given a list of possible alternatives – a kind of lookup or thesaurus facility. Figure 2 shows the XML definition for an extract of the Airport example. The classes might include Terminal (Building), Airline (Carrier, Company), Flight, and Runway (Landing) with alternative synonyms shown in brackets. In a similar way, possible alternatives to attribute types and method names are also identified by the tutor.

```
<Airport>
  <class name="Flight">
    <equivalent>Plane</equivalent>
    <equivalent>Aeroplane</equivalent>
    <equivalent>Plain</equivalent>
    <attribute name="pilot" type="Pilot">
      <equivalent>Captain</equivalent>
      <equivalent>Flyer</equivalent>
    </attribute>
    <attribute name="passengers" type="Passenger[]">
      <equivalent>Seat[]</equivalent>
      <equivalent>Vector</equivalent>
      <equivalent>ArrayList</equivalent>
    </attribute>
    <method name="setPilot">
      <attribute name="pilot">
    </method>
    <method name="getPilot">
      <return-type name="pilot">
    </method>
    <method name="setPassengers">
      <attribute name="passengers">
    </method>
    </method>
  </class>
</Airport>
```

Figure 2: Much abbreviated equivalence MAP

As students submit project work the final designs are analyzed and used to refine the equivalence maps (largely by adding synonyms). In certain cases, sets of completely different, yet successful designs are submitted and in these cases, new equivalence maps are developed.

Using the current version of VorteX, students are given their specification and are presented with an empty screen on which to start their design. At various stages of development they can ask the system for assistance. The idea of an assistant provides a means to aid a lost student. By providing the student with a suggestive aid we aim to inspire and guide them rather than channel students' designs in a similar direction, removing their unique perspective. Exactly what is presented is dependent on what the instructor wishes to make available. If there are no restrictions, students might select the highest-level view that shows the classes that have been used or they might zoom-in

to see the individual methods defined within the classes. The instructor is able to alter what is available through the lifetime of a project. It might not be desirable to give out the full solution at the start of the development; it might start with just the basic class outline followed some days later by a more detailed breakdown.

If the student has already produced an initial design, they can at any stage ask the system to rate their work. By comparing the current design with those within the case base, VorteX will give information on the closest match. It will respond with information such as "Your design currently has a close match (85%) with the design chosen by 63% of students".

10. CONCLUSION

We hope that the initial case-based system for software designers will prove a real asset in teaching software development by improving the quality of the educational experience. Most Computer Science Departments are only too aware of the low success in helping novice programmers develop their programming and design skills. This system will assist in speeding up the learning process for the learner by helping them gain knowledge that is usually only developed through extensive experience.

Once the case-based system has been finalized, VorteX will move into the final most adventurous phase, which is to take the design cases and factor out a more generalized model of the software design learning process thereby breaking out of the confines of the original case studies. We expect to be able to abstract a set of general causal mechanisms appropriate to novice programmers from the case studies and from the experience with the case-based system. We will then apply these to new case studies to generate for unseen problems the kind of support that the case-based system provides for the original case study problems. Stroulia does similar abstraction and reuse of design principles, albeit in a simpler domain [14].

This final stage of the project aims to develop and refine a generalized model of the learning process facing novice designers and identify effective techniques that can be used to help overcome them.

11. REFERENCES

[1] Ratcliffe, M.B. Improving the Teaching of Introductory Programming by Assisting the Strugglers. The 33rd ACM Technical Symposium on Computer Science Education, Cincinnati, USA, March, 2002.

[2] Ratcliffe, M.B., Woodbury, J and Thomas, L.A. A Pedagogically Driven, Directed Learning Environment, 2nd Annual LTSN-ICS Conference, University of London, August 2001.

[3] McCracken, M., et al., "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students", report of an ITiCSE 2001 Workshop, SIGCSE Bulletin, December 2001.

[4] Maris, J.M., and VanLangen, C. A Design Tool for Novice Programmers, Working Paper Series 00-01-April 2000, http://www.cba.nau.edu/working_papers/papers &abstracts/ MarisVanLanLucy/Novice.htm

[5] Arnow, D., and Weiss, G. Introduction to Programming Using Java: An Object Oriented Approach, Addison Wesley, Menlo Park, California, 2000, p. 142.

[6] Booch, G., Object Oriented Design with Applications, Benjamin/Cummings, Colorado, 1991, p. 190.

[7] Rumbough, J., et al., Object Oriented Modelling and Design, Prentice Hall, Englewood Cliffs, New Jersey, 1991, p. 47.

[8] Coad, P., and Yourdon, E., Object Oriented Analysis, 2nd edition, Yourdon Press, Englewood Cliffs, New Jersey, 1991, p. 48.

[9] MacGregor, M., Thomas, P., and Woodman, M. AESOP (An Electronic Student Observatory Project), ITiCSE 2001, Innovation & Technology in Computer Science Education, Canterbury, Kent.

[10] Holmer, T. and Schummer, I., A Tool for Co-operative Program Exploration, ECOOP 2000 Workshop, Tools and Environments for Understanding Object-Oriented Concepts, June 12, 2000

[11] Silver, E.A., Foundations of cognitive theory and research for mathematics problem solving instruction in A. H. Schoenfeld (Ed.), Cognitive Science and Mathematics Education, Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 33-61.

[12] Beck, K., Extreme Programming Explained: Embrace Change, Addison-Wesley, USA, 2000.

[13] Thomas, L.A., Ratcliffe, M.B., and Robertson, I.A., Code Warriors and Code-a-phobes: A Study in Attitude and Pair Programming, Submitted to SIGCSE 2003.

[14] Goel, A. and Stroulia, E., Functional Device Models and Model-Based Diagnosis in Adaptive Design, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol 10, 1996, pp. 355-370.