Understanding our Students: Incorporating the Results of Several Experiments into a Student Learning Environment

Mark B. Ratcliffe and Lynda A. Thomas
Department of Computer Science,
University of Wales, Aberystwyth, Wales, UK
{mbr,ltt}@aber.ac.uk

Keywords: POP-I.A. learning to program, POP-I.B. barriers to programming, POP-IC, educational technology, POP-II.A. learning styles, novices, POP-II.B. program comprehension, POP-IIID. editors.

Abstract

This paper describes a project that aims to enhance student learning of Object Oriented Programming through the development of an interactive learning environment. Through a series of connected experiments we have sought to discover more about how our students learn and have incorporated this knowledge into a specially tailored Integrated Development Environment (IDE), VorteX.

Introduction

For several years, researchers (Robins et al., 2003) have been concerned that beginning students of computer programming have not been performing to the levels expected by their instructors. In an international study organised by McCracken (McCracken et al., 2001) of first year programming students who were attempting to solve an assigned programming problem, the average grade was just 21%.

Perhaps because we are, after all, software professionals, CS educators often look for a technical solution to this and other problems. Instructors of most programming courses, with the intention of assisting the learning process and reducing the workload, thus use IDEs on a daily basis to aid in their teaching. Almost all these IDEs, however, are designed for practitioners in the field rather than beginners. The exception to this is BlueJ (Kolling & Barnes, 2002), which, while it has many excellent features, does not provide all the support that we, at Aberystwyth, felt was required.

In particular, features that we wished for our own students' environment were:

- support for scaffolding of key concepts, and
- support for collaborative learning.

In addition, we were very aware that our students gave us their finalised designs for programming projects, but we had little information about how they arrived at these designs. Therefore, another primary force in the inception of our student IDE was that we wanted:

- a way of collecting information on how our students actually approached the design and coding process.

In order to address these concerns, we decided to build our own IDE, but we wanted to be sure that the design would be based on real student (not just instructor-perceived) needs. We also wanted the IDE to be part of an on-going and evolving project rather than the static product of it.

In this paper we go on to describe a series of experiments that were undertaken in order to understand our student needs. In addition, we describe the effect of our findings on the production of a better foundation for the design of the evolving IDE, which is currently called VorteX. We conclude with some more general points on our experience of trying to learn more about our students and their learning needs.

An Experiment with Learning Styles

There have been many characterisations of learning style preference. One theory, first popularised in the 1960s, claims that each brain hemisphere is specialised for a different mode of thinking: the left for linguistic, analytical and sequential tasks and the right for artistic, gestalt and creative tasks (Rohr, 1986). Another way of looking at learning examines it from the perspective of Visual, Auditory and Kinaesthetic/Tactile preferences. Variations of this analysis have, for example, been applied in the context of learning English as a second language (Reid, 1998).

Richard Felder has studied different student learning styles in the context of Chemical Engineering. Felder's writings (Felder, 1996) provide both a valuable overview and an extension of the work in this area, and also prescribe an approach to education that incorporates the recognition of different learning style preferences. Felder categorises learners as preferring: visual/verbal, active/reflective, sensing/intuitive, sequential/global and inductive/deductive styles.

What appealed to us was that the emphasis in Felder's work is on *preferred* learning style, not some kind of hard-wired and immutable characteristic. He notes that:

> "Functioning effectively in any professional capacity, however, requires working well in all learning style modes. … If professors teach exclusively in a manner that favors their students' less preferred learning style modes, the students' discomfort level may be great enough to interfere with their learning. On the other hand, if professors teach exclusively in their students' preferred modes, the students may not develop the mental dexterity they need to reach their potential for achievement in school and as professionals. An objective of education should thus be to help students build their skills in both their preferred and less preferred modes of learning." (Felder, 1996).

Felder claims that for at least the past decade, engineering instruction has been biased heavily towards the intuitive, verbal, reflective, sequential learning styles, yet few students fall neatly into all of these categories.

Whichever model of learning styles one favours or accepts, Felder's crucial observation for us was that there is and has been a profound mismatch between learning and teaching style preferences. Such a mismatch may result in student disinterest, apathy and underachievement. He therefore advocates teaching strategies that incorporate all of the learning style preferences.

There is a certain amount of controversy about whether the theory of preferred learning styles has validity (Coffield, 2004). But there *is* general agreement that reflection is of considerable utility to students (Tanner & Jones, 1999); and the point that technical teaching has tended to favour the preferred modes of the instructors rather than the students is a crucial one.

To investigate whether students with particular learning styles were performing better than others, we undertook an experiment. that concluded that judging by student performance, we were probably favouring students with certain style preferences in our instruction {Thomas et al., 2002). In particular, active and visual learners did not do as well in our introductory programming sequence as other students. These students are identified by Felder as responding to trying things out and working with others and to prefer pictures, diagrams and flow charts to written or spoken explanations.

This information has been reflected in the design of the VorteX environment in that students are encouraged to reflect on their preferred style of learning and can choose from a wide range of supported activities (lectures, exercises, group work etc.).

The Knowledge Barrier

In this section let us note that there is a related problem of instructors imposing their own 'world view' on students. This is particularly obvious when inexperienced advisors/tutors suggest alternative approaches to confused learners rather than explaining what is really wrong with their designs. Novice developers are often left wondering why their quite reasonable designs were 'wrong'. Unfortunately the tutors' experience puts up a significant barrier and can prevent the tutor from appreciating the real difficulties that the learners face. This difference in experience represents the 'knowledge barrier'; the problem is how to break this down.

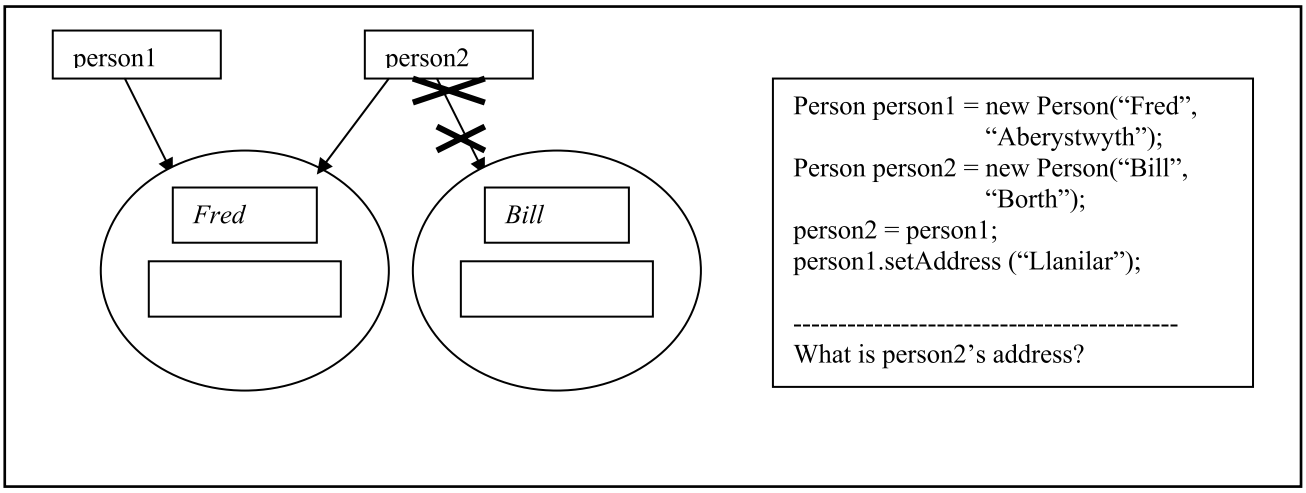
To further demonstrate how this knowledge barrier obscures student learning, research suggests that novices have greater difficulty using available help than the more experienced (Allwood, 1986). They often have difficulty in formulating questions and in understanding the advice given. Further research indicates that experts focus more on functional relationships and beginners on object and class relationships (Davies, 1995). As Davis notes, "Students construct knowledge by making conjectures that become rules. This process fails if students abandon their own efforts and instead rely on pronouncements from authorities." (Davis, 1993).

There is a difficult balance to be maintained here. Research has shown that methods of teaching and learning that combine significant student autonomy with dynamic scaffolding by the teacher are indeed highly effective (Tanner & Jones, 1999). So, how much help is too much, and what techniques can we give to our students that will help them the most in coming to grips with design and coding?

An Experiment with Scaffolding and Visual Learning material

After discovering that our students might benefit from visual materials, and with the design goal of supporting scaffolding in mind, we turned to consider how VorteX might help students visualise programming ideas.

Many students seem to have considerable difficulty understanding program behaviour and one of the manifestations of this lack of understanding that we have observed in our classes is that many of them do not seem to be able to trace code. In particular, in the context of Object-oriented programming, many students are not able to produce (even informally for themselves) a diagram that demonstrates an understanding of object references.



```
Person person1 = new Person("Fred",
                 "Aberystwyth");
Person person2 = new Person("Bill",
                 "Borth");
person2 = person1;
person1.setAddress ("Llanilar");

------------------------------------------
What is person2's address?
```

Figure 1. A partially completed Object diagram and related question[1]

Holland et al. regard the misconceptions around object references as one of the most fundamental in learning Object-oriented programming and suggests that "the cleanest way to defuse this misconception is to teach reference as a first class concept …" (Holland et al., 1997).

[1] Note that this diagram only provides the flavour of the type of question. The actual questions involved two classes that were previously well known to the students and came with some basic code. In addition we are 'cheating' in the representation of the Strings, which are of course really references not primitive values. The idea was to make the diagrams 'sloppy' but effective.

In the introductory programming sequence in Aberystwyth, we certainly try to do this, demonstrating program behaviour in lectures and tutorials mainly by drawing pictures of variables and what they reference, as in Figure 1 (essentially rough Object (Instance) diagrams (Fowler & Scott, 1999)). So students *have* been exposed to the idea of tracing through Object diagrams – but when they might appropriately use it themselves, weaker students fail to do so. Even in the more restricted situation of a test that asks the students to trace out what happens when just a few lines of code are executed, as in Figure 1, scratch sheets are often returned blank. In a previous investigation we discovered that only 36% of the sheets were returned with any kind of 'working out' on them (Ratcliffe et al., 2003).

We conjectured that providing scaffolding (in the form of partially completed Object diagrams) would help our students understand the concept of object referencing when tracing test code, and also encourage them to use Object diagrams in tracing their own code. We intended to incorporate the feature of helping students visualise program behaviour with respect to object references into the VorteX environment, but first we conducted an experiment using paper copies of partially completed object diagrams with students on multiple choice test questions.

Specifically, we focused on three questions:

- Is drawing some kind of Object diagram correlated with success in solving multiple-choice tracing questions?
- Does providing students with scaffolding in the form of partially completed Object diagrams help them correctly answer multiple-choice tracing questions?
- Do students who have been provided with this scaffolding continue to use it in such multiple-choice questions?

In the first question we were looking at diagrams that the students may draw with or without our encouragement. With respect to the second question, we had to consider what we mean by 'help'. If we give students a piece of code and an incomplete Object diagram it seemed very unlikely that they would NOT do better than if we simply give them the code with no help whatsoever (but see below). We wanted to confirm this conjecture but were originally, in fact, more interested in research question 3: whether giving the students incomplete Object diagrams in their early learning would be helpful to their later ability to trace code. We conjectured that it would – and our intention was to then incorporate this facility into VorteX.

We were very surprised by the results of this experiment (Thomas et al., 2004). There was some evidence that, for beginning students, drawing Object diagrams is in fact correlated with understanding object references and being able to trace code. But *providing* these students, with what we considered to be helpful diagrams, did not significantly appear to improve their understanding, as measured by performance on tracing questions. This was completely unexpected. We thought that we were "practically doing the question for them", as one of our lab assistants remarked, and expected that the experimental group would do brilliantly on the tracing questions. In fact, these students did not appear to gain much from the partially completed diagrams at all.

We were influenced in the foundations of this experiment by the work of Mary Hegarty who has made a study of cognitive models that impact on the understanding of mechanical systems. In the paper by Narayanan and Hegarty (2000), the authors outline a cognitive model of understanding dynamic systems that supposes that the viewer:

- decomposes the system into simpler components,
- constructs a static model by making representational connections to prior knowledge and other components,
- integrates information between different representations (e.g. text and diagrams),
- hypothesizes lines of action, and finally
- constructs a dynamic mental model by mental animation.

The design guideline that people learn more from being induced to mentally animate a system before viewing an animation than by passively viewing the animation has also been empirically validated (Hegarty et al., 2002).

We note that by *providing* the students with the diagrams we were perhaps removing the first few steps of the Narayanan/Hegarty model of understanding systems. Maybe it is necessary for the students to build the Object diagram (model) themselves in order to animate it. Scaffolding might be a useful idea – but we may have inadvertently provided the wrong kind of scaffolding, or the wrong level of scaffolding.

In any event, this experiment, with its somewhat surprising and disappointing results, convinced us that adding scaffolding with Object diagrams to VorteX (and later animation of those diagrams as we had initially planned) was not appropriate at this point. This does not mean that scaffolding itself is a mistake and we are investigating other kinds of scaffolding at present (see section 7).

Collaborative Working

There is considerable evidence that collaboration can have a very positive impact on learning (Webb & Peterson, 1992), (Swing, 1982). In a study by Swing and Peterson, it was reported that students of low achievement benefited from working in groups that were composed of mixed ability rather than a group made of homogenously low achievers. The benefit comes from struggling students being able to feed off the knowledge of the more able students. This knowledge barrier (as discussed in section 3) between group members is less than that between student and lecturer. The theory is that a student with a higher than average understanding of their course, but at the same stage (*i.e.* started the course at the same time) will have recently overcome the problems and misunderstandings of the struggler and can perhaps more easily understand, empathise and explain.

Our student learning style profile also suggested that we should add more opportunities for group work, and our own experience led us to believe that many students learn better when working with others. In fact our instinct was to base VorteX around collaborative learning, but we also wanted to validate this idea. Would it really work with our own students?

An Experiment with Pair Programming

One place where collaborative work is utilised is in the practice of pair programming. Pair programming is “a programming technique where two people program with one keyboard, one mouse and one monitor” (Beck, 2000). It has gained a good deal of interest recently, probably because of its inclusion in the set of techniques that comprise Extreme Programming. Initially the evidence for the effectiveness of pair programming was largely anecdotal, but recently surveys in industrial (Williams et al., 2000) and educational (McDowell et al., 2003), (Williams & Kessler, 2001) settings seem to indicate that most respondents find it enjoyable and that it improves the level of code quality.

As educators, we wanted to see if these results could be duplicated with our own programming students. We could certainly believe that this practice might work well with mature programmers in industry, but we had certain niggling doubts as to how well it would work with 18 year olds straight from high school.[2] We were particularly concerned in that amongst our students we had observed two particular minority subgroups whom we identified as ‘code warriors’ and ‘code-a-phobes’. This appeared to be partly a matter of ability, but even more a matter of attitude.

In this experiment (Thomas et al., 2003) we asked students to self identify on a code- warrior – code-a-phobe scale and we then placed the ones who appeared to be at the extremes of the scale first with opposite, and later with like, pair programming partners (the middle group of students were placed with each other.) We discovered that:

[2] The origin of these doubts was Dr. Williams’ description of the average student at the University of Utah. Many of these students have postponed University while they completed a missionary year. In addition, many are married. This is quite different from the profile of most of our students, who are considerably less mature and not used to cooperative endeavour.

- Overall, our students liked pair programming and believed that it helps them achieve good solutions.
- Students with less self-confidence seemed to enjoy pair-programming the most[3].
- The students whom we identified as ‘warriors’ liked pair programming the least. This was as we suspected given our student profile.
- There was some evidence that warriors like pair programming even less when they are paired with phobes and that students produce their best work when paired with students of similar, or not very different, levels of confidence.

Thus, we confirmed that our students, on the whole, liked and seemed to learn more when they could work in groups and that Swing and Peterson’s theory of mixed ability groups had some (although not complete) validity for our students. We continue to use pair programming in our labs and encourage students to use it out of formal lab time, but often students want to work in physically separate locations.

Obviously, pair programming has more to it than just the fact that the students are working collaboratively: the fact that one student is taking the larger view, the fact that the students are sharing a monitor, etc. We could not replicate this experience directly in a distributed environment. But the pair programming experiment gave us confidence that our design goal of basing VorteX around support for collaborative working was valid. It also gave us ideas about how we wanted distributed collaboration to work. In particular, we wanted to encourage the pair programming idea that pairs ‘talk’ about what they are doing and why.

An Experiment with Collaborative working in VorteX

There are, however, difficulties associated with group working, particularly in an educational setting. Evaluation of the performance of individuals within a team is difficult (Sheridan et al., 1989). This is especially true when a group is working toward a single goal (a disjunctive task) without any individually assigned tasks, as the only assessable material is the end product. Unfortunately the harder working students often feel they are being unfairly evaluated. (This was brought up by some of our ‘code warriors’ as a reason why they did not like pair programming.) Without close supervision, it is possible for some members to do little work and rely on their team members to complete the tasks. “When group members realise that their efforts are dispensable (group success or failure depends little on whether or not they exert effort), and when their efforts are costly, group members are less likely to exert themselves on the groups behalf” (Johnson & Johnson, 1994).

Slavin (Slavin, 1990) has suggested three qualities that collaborative learning projects must have if they are to succeed, namely team reward, individual accountability, and equal opportunities for success. If these qualities can be addressed within an environment to support group working then we hoped that the disadvantages of collaboration could be surmounted whilst retaining all of the advantages.

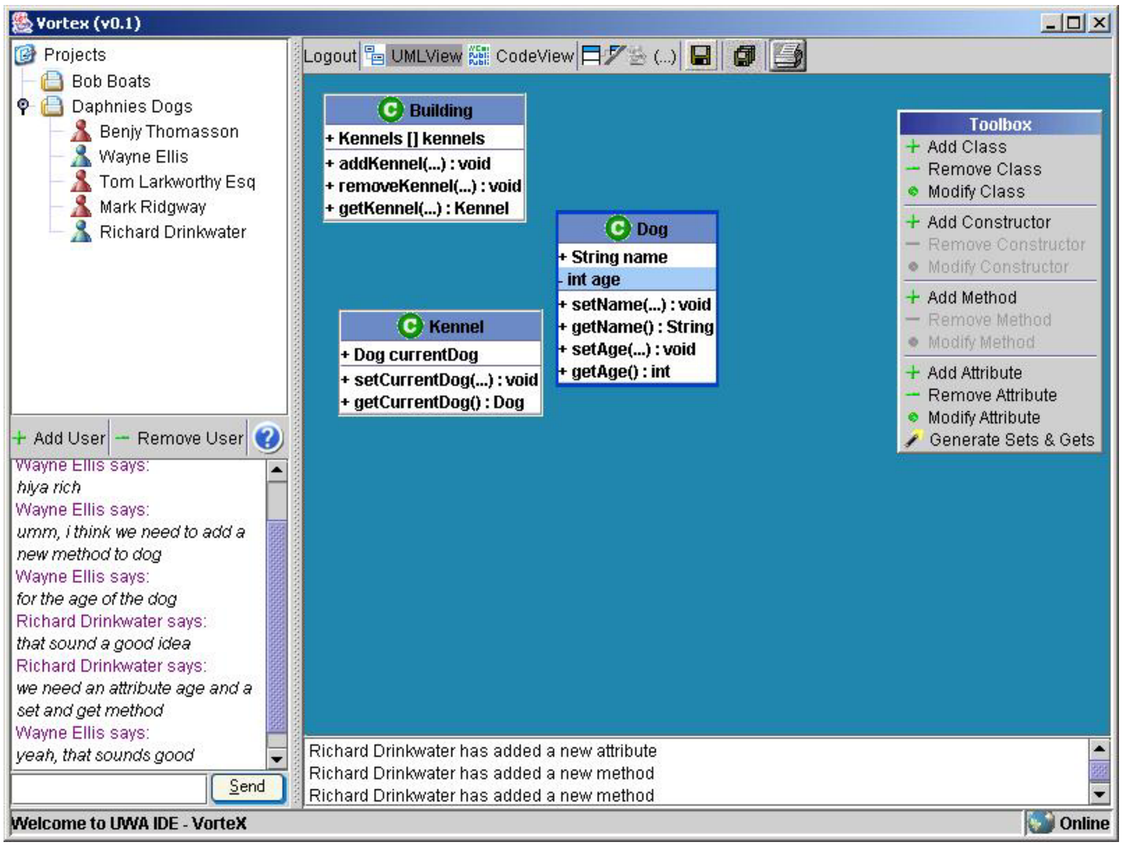
As mentioned previously, rather than just theorise about how students design programs, we also wished to be able to make observations about that process. The VorteX tool has been constructed to capture that information by requiring users to explain their actions at every stage of the design process. These explanations give a three-fold benefit: they generate in-line documentation; they force the users to think about each action they are performing; and they provide the instructor with valuable data about student misconceptions. This process can happen in single user mode but is most useful in a collaborative context.

[3] An interesting spin off from this experiment is that some students in the second year of our program adopted pair-programming of their own accord for dealing with less confident group project members.

VorteX

This experiment is different from the others on which we have reported, in that, obviously, we could not test the features discussed above without actually *using* the tool. A preliminary version of VorteX is indeed available and was used in this experiment.

VorteX provides a fully interactive, collaborative development environment that enables the capture of novice design knowledge and gives structured feedback to educators. The most significant idea behind VorteX is that it monitors the students as they design their software systems. At each step, as 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. Collaboration between group members in VorteX is facilitated through the environment's built in chat tool. Messages sent between users are time stamped and logged for future analysis.



Wayne Ellis says:
hiya rich
Wayne Ellis says:
umm, i think we need to add a new method to dog
Wayne Ellis says:
for the age of the dog
Richard Drinkwater says:
that sound a good idea
Richard Drinkwater says:
we need an attribute age and a set and get method
Wayne Ellis says:
yeah, that sounds good

```
Richard Drinkwater has added a new attribute
Richard Drinkwater has added a new method
Richard Drinkwater has added a new method
```

Figure 2: VorteX - student user view

Figure 2 shows a snapshot of the tool for a student user working on a group project. The top left area of the window shows the available projects for that user. To encourage collaborative design, VorteX enables students to be designated as belonging to one or more project groups. These teams are fundamental to VorteX and represent all of those members who have access to a particular project.

The selected project shows five group members with only 2 of those members currently active. The lower left window shows the chat window that allows entry of free text. All communication within this window is tagged to developments within the main design area, making it available at a later stage to help justify changes that were undertaken. The main design area shows a number of classes that have been created in the project. Individuals are presented with their own view of the design that can be diagrammatic or textual. VorteX supports the Universal Modelling Language (UML) (Fowler & Scott, 1999) for representing classes as diagrams. Any changes made to either the textual or diagrammatic view have to be justified, and are then transmitted to the whole group in real time.

Using the main VorteX tool, student members can, at any time, view the current project status. The real strength of VorteX lies in what can be done with its stored information. Using an associated animator, individual team members (and their instructors) can view the entire build up of the design showing each action step by step. Team members who may have missed a session or two are now able to quickly catch up on recent developments and, in a similar way, groups are able to carry out reviews of recent changes.

A similar web-based administration interface can be used by instructors to provide snapshots of the current designs as they are progressed by each student group. This facility enables an instructor to quickly assess the relative progress of the individual teams; identifying which, if any, need specific attention. Once an individual team has been selected, extensive information can be provided to help analyse both the group's progress and individual contributions. Such information would be laborious to obtain by manually analysing the accompanying history logs, but the customisable metric software available through the administration tools enables accurate identification of individual contributions to the group work.

The information stored by Vortex can be used by instructors and researchers to build a picture of the team's progress, the individual contributions and also student understanding of programming concepts through an examination of the associated discussions that were being held at the time of changes.

The Experiment

VorteX has been used in earnest at Aberystwyth for the last six months and the results are already very encouraging. After extensive trial testing it was decided to introduce the tool to a group of 100 freshman students each of whom had only four weeks of programming experience. We wanted to see how well they managed with the completely unfamiliar tool on an unseen design problem.

Once students had downloaded and installed the tool, they were assigned a unique user-id to allow them access to VorteX. Efforts were made to ensure that students were not assigned to the same group as the person sitting next to them. In this way each student within a group of four was completely unaware of with whom they were collaborating. Although the experiment was not intended primarily to evaluate anonymous working it provided some very interesting results in this area.

The students were asked to produce a top-level design for a system of several inter-relating classes. For them this was a challenging but attainable task. Using VorteX, the groups of individuals were able to develop their applications by collaborating remotely in real time. Using the shared diagram editor each member within the group was independently able to manipulate either UML diagrams or Java code (whichever they preferred) adding new classes, methods and attributes and seeing whatever changes their colleagues made as they happened. They had to justify their actions as outlined above.

Following the experiment students were asked to complete a questionnaire, the results of which are typified by the following extract:

- “I found the experience both enjoyable and insightful. Initially I was a little apprehensive at not knowing who I was working with. Vortex was pretty intuitive and easy to use. What I really found interesting about the experience was that it was almost like plugging straight into my colleague's heads, in effect bypassing or filtering out all the visual "noise" associated with day-to-day communication. This permitted us to operate mainly on an intellectual level. Working in this way allowed Student-X [*a profoundly deaf student*] to interact with us on an equal footing without the usual communication problems and physical barriers enabling him to work freely and express his ideas effortlessly. I felt that the anonymity Vortex provided, was helpful in the sense that if there was anything one of us did not understand, or had not come to grips with during the lectures, we could express it freely and get help from one another [Suzana Maria Barreto].”

Other replies from the questionnaire include:

- “It made it easy, I could make stupid mistakes anonymously. It was also easier to talk (or write messages in this case) when not face to face.”
- “Interactive - gave me lots of ideas that I didn't think of.”

- “Very good, improved my communication with people I didn't know”
- “Easier to remove syntax errors by working as a team.”
- “Everyone can put their point across without saying too much, *e.g*. Some people can be quiet when with 'real' people.”
- “If a decision is reached about two different ways of doing something and the group is split on the matter, it is more difficult to resolve, but being anonymous makes it easier to say what you need to.”

It was interesting to note that although many students concluded the session by finding out who they had been working with, 40% of the participants made no effort to identify their colleagues, even though nothing was said to either discourage or encourage this. Only one team attempted to identify their colleagues at the start of the exercise. Within minutes of starting VorteX, this team messaged all members to meet at the back of the room. Interestingly, for them the experience was less positive. Every other group rated the exercise as 'very positive', convincing the authors that anonymous working has much to offer the learning process and is certainly worth pursuing.

Continuing Experimentation

The ability of VorteX to monitor the actions of all users within the tool and give extensive statistical feedback helped to address many of the problems of collaborative learning and working discussed previously. There is a team reward, but also individual accountability and equal opportunity as Slavin suggested.

The logs of the final designs have been collated and analysed (Thomasson et al., 2003) to look for patterns, with the aim of gaining a greater understanding of the student learning process. This collected information is allowing us to better see how students think. We can step through their design action-by-action looking at how their design was built, following their conversations and justifications for their actions. In short, we are getting a greater insight into how students perceive Object Orientation and obtaining a better understanding of their grasp of the concepts.

In addition, we have at least a handle on the knowledge barrier, which was discussed in section 3. Now if a lab assistant or instructor makes a suggestion that radically changes student designs, there is a trail that will be clear in the animation. This makes it more obvious where students perhaps needed more subtle help or where their understanding should have been better established in advance of the project.

Learning about our students

As we reflect on the last few years in the development of Vortex we are pleased to note that most of its features have been developed as result of real and tested student needs. We believe that we have not fallen into the trap discussed by Felder of teaching entirely from our own perspective – but rather we have tried to discover more about our students and progress from that starting point. This is evidenced not just in the development of VorteX , but also in an approach to teaching that is student centred.

Of our three original design goals (support for scaffolding of key concepts, support for collaborative learning and a way of collecting information on how our students actually approached the design and coding process) we have accomplished the last two. As described in section 4, we considered a situation in which we could incorporate the first goal but discovered that, at least as we then constructed the scaffolding, it would not be likely to be helpful to our students. It was quite difficult to give up (at least temporarily) something that we considered such a ‘good idea’ and we will probably revisit it, but it was important to the process of the team that we build VorteX from a student-tested focus.

We are currently working on another kind of scaffolding for VorteX – that is an intelligent tutoring system that is based on case-based reasoning (Thomasson et al., 2003). Very simplistically, by identifying the commonalities in student designs, we have been able to simplify the design structures that are represented in our case-based system. Current designs can now be pattern-matched against those from previous years and their authors given suggestions on how to improve their designs.

This experimentation in order to discover what our students really need is on-going and will doubtless lead to refined versions of VorteX in the future.

References

Allwood, C.M. (1986). Novices on the computer: a review of the literature, *International Journal of Man-Machine Studies*, *25*, 633-658.

Beck, K. (2000). *Extreme Programming Explained*, Addison-Wesley (2000).

Coffield, F. (2004). Revealing figures behind the styles, *Times Higher Education Supplement,* Jan 2nd 2004, 20.

Davies, S.P. (1995). Are objects important? Effect of expertise and familiarity on the classification of object-oriented code, *Human-Computer Interaction, 10,* 227-248.

Davis, E.A. (1993). Mind your Ps and Qs: using parentheses and quotes in LISP, in C. R. Cook et al., (Eds.), *Empirical Studies of Programmers: Fifth Workshop*, Norwood, NJ: Ablex, 62-85.

Felder, R. M. (1996). Matters of style, *ASEE Prism,6* (4), 1996.

Fowler, M., & Scott, K. (1999). *UML Distilled: A Brief Guide to the Standard Object Modelling Language,* Addison Wesley.

Hegarty, M., Naranayan, N.H., & Freitas, P. (2002). Understanding machines from multimedia and hypermedia presentations. in J. Otero, A. C. Graesser & J. Leon (Eds.). *The Psychology of Science Text Comprehension*. Lawrence Erlbaum Associates.

Holland, S., Griffiths, R., & Woodman, M. (1997). Avoiding object misconceptions. *Proceedings of the Twenty Eighth Technical Symposium on Computer Science Education, SIGCSE 1997*, ACM Press.

Johnson, D. W., & Johnson, F. (1994). *Joining Together: Group Theory and Group Skills*, 5th Ed, Englewood Cliffs, New Jersey: Prentice Hall.

Kolling, M., & Barnes, D. J. (2002). *Objects First with Java: a Practical Introduction Using BlueJ: An Introduction to Object Oriented Programming*. Englewood Cliffs, New Jersey: Prentice Hall.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A multi national study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin. 33*(4), 125-140.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair programming on performance in an introductory programming course, *Proceedings of the Thirty Third Technical Symposium on Computer Science Education, SIGCSE 2002*, ACM Press.

Narayanan, N. H., & Hegarty, M. (2000). Communicating dynamic behaviors: are interactive multimedia presentations better than static mixed-mode presentations? in Michael Anderson, Peter Cheng and Volker Haarslev (Eds.), *Theory and Application of Diagrams, Diagrams 2000*, , Springer Lecture Notes in Artificial Intelligence 1889.

Ratcliffe, M.B., Thomas, L.A., Ellis, W., & Thomasson, B. (2003). Collaborative designs to assist the pedagogical process. *The 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2003)*, Macedonia, Greece, ACM Press.

Reid, J. (1998). *Understanding Learning Styles in the Second Language Classroom*, Prentice Hall.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: a review and discussion, *Computer Science Education*, *Vol. 13, Number 2.*

Rohr, G. (1986). Using visual symbols. In S.Chang (Ed.) *Visual Languages*, Plenum, 1986.

Sheridan, J., Byne, A.C., & Quina, K. (1989). Collaborative learning notes from the field. *College Teaching.* Vol 3. 37(2), pp. 49-53.

Slavin, R. E. (1990) *Cooperative Learning: Theory, Research and Practice.* Needham Heights: Allyn and Bacon.

Swing, S., & Peterson, P. (1982). The relationship of student ability and small group interaction to student achievement. *American Educational Research Journal. 19,* 259-274.

Tanner, H., & Jones, S. (1999). Dynamic scaffolding and reflective discourse: the impact of teaching style on the development of mathematical thinking, *Proceedings of the 23rd Conference of the International Group for the Psychology of Mathematics Education,* Haifa, 1999.

Thomas, L.A., Ratcliffe, M.B., & Woodbury, J. (2002). Learning styles and performance in the introductory programming sequence. *Proceedings of the Thirty Third Technical Symposium on Computer Science Education, SIGCSE 2002*, ACM Press, 33-37.

Thomas, L.A., Ratcliffe, M.B., & Robertson, I.A. (2003). Code warriors and code-a-phobes: a study in attitude and pair programming. *Proceedings of the Thirty Fourth Technical Symposium on Computer Science Education, SIGCSE 2003*, ACM Press.

Thomas, L.A., Ratcliffe, M.B., & Thomasson, B.J. (2004). Scaffolding with object diagrams in first year programming classes: some unexpected results. *Proceedings of the Thirty Fifth Technical Symposium on Computer Science Education, SIGCSE 2004*, ACM Press.

Thomasson B. J., Ratcliffe, M.B., & Ellis, W. (2003). Eliminating redundancy from novice design information in a case-based reasoning system. Submitted to *Advanced Engineering Informatics* (2003).

Webb, N.M. (1992). Task related verbal interaction and mathematics in small groups. *Journal for Research in Mathematics Education*, 22, 366-389.

Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair-programming, *IEEE Software* (July/Aug 2000).

Williams, L., & Kessler R. (2001). Experimenting with industry's "pair-programming" model in the computer science classroom, *Journal of Computer Science Education* (March 2001).