Students Asking Questions: Facilitating Questioning Aids Understanding and Enhances Software Engineering Skills

David Barnes
The Computing Laboratory
The University
Canterbury, Kent. CT2 7NF
United Kingdom.
d.j.barnes@ukc.ac.uk

Abstract

By providing a means to ask questions anonymously, we provide a non-threatening environment in which students are encouraged to fully understand and criticise their assignments. As well as providing practical assistance for those who are struggling with course work, it demonstrates a practical reinforcement of ideas that are commonly taught in courses on software engineering, but it does so in a context that they can directly relate to – their desire to achieve good grades. The approach is based around the use of an HTML form to enable the anonymous submission of questions to staff and the dynamic refinement of assignment specifications. Examples of its use in a course on object-oriented design and C++ are given.

1 Motivation

How often do we look at a piece of course work and reflect that the student submitting it clearly did not have the first idea how to start it, or had misunderstood an essential part of what they were being asked to do? 'Why didn't they come and ask for help?' is our frustrated cry! Poor work is often simply the result of a failure to understand what is required. If we can find ways to overcome that lack of understanding, we may be able to greatly increase the chances that our students will gain the fullest benefit from the assignments we set, as well as other course material. Experience shows that some students are simply afraid to ask for help – too worried about what we might think if they ask 'a stupid question'. Some, indeed, worry that they will lose marks if they ask for help, and prefer to struggle on in ignorance rather than risk that possibility. The setting of assignments is driven not just by the need to produce assessment grades, but to enhance the learning process by providing reinforcement of taught course material. This work was motivated by the desire to enable students to ask more easily for help so that they might get the fullest benefit out of the assignments being set for them.

2 Emphasising Understanding

A significant amount of the mainstream Computer Science teaching at the University of Kent involves relatively large software assignments, where the principles and practices of software engineering are taught and expected to be adhered to. A fundamental part of the teaching of software engineering is to emphasise the need for critical evaluation of artifacts throughout the process.[7, 12] Nothing should be taken as given, but should be critically analysed for incompleteness, contradiction and unnecessary constraint. We teach that this process must be applied to initial requirements, formal specifications, designs and implementation. We do this because such an attitude of criticism promotes understanding in the minds of those performing it and identifies weaknesses in the work of others that benefit from early identification. We teach that it is very difficult to specify systems completely and unambiguously.

Within the main programming components of the first year of our Computer Science major programme, we have begun to emphasise the primary role of understanding through the use of a problem solving technique called 'How to Program It' [3], which is based upon Polya's 'How to Solve It' [10]. The four components are Understand, Design, Write and Review. This is described as a cycle with the Review phase feeding back into the Understanding phase of future problems. By placing Understanding as the starting point, we emphasise its key role in dealing with all tasks that they are set, and that they should never try to design a program until they are sure that they have understood what the assignment is asking them to do. The software engineering experience proves that the need for this stage is not always appreciated!

Our problem solving approach is really a lightweight version of the approach that is common to most courses on software engineering – requirements elicitation and analysis, specification, design, implementation, and legacy capture. It is widely accepted that requirements analysis is an essential precursor to design, in order to ensure that the requirements are complete, consistent, and not unnecessarily constraining of implementation. To embark upon design before the system has been fully understood and accurately specified is asking for trouble. Yet, do we apply such a single-minded attitude to the assignments that we set our students? Do we provide them with the means to critically evaluate and refine the tasks we set them, or do we (unconsciously, perhaps) imply that such theory has no real application to the tasks that need doing today?

3 An Anonymous Question Asking Form

The initial idea for this attempt to make it easier for students to ask questions came from an alternative approach employed by one of my colleagues with her non-specialist computing students. As I walked into the lecture theatre to take my class, I noticed that her students were dropping pieces of paper into a box as they left. Her key was the guarantee of anonymity, and this was her way of enabling them to ask questions anonymously. I realised what a brilliant way this could offer to reach my students, but pieces of paper are part of an alien culture to CS majors, so I turned to the Web!

The anonymous question asking form makes use of a feature provided by most Web browsers - anonymous browsing. In the main, when you visit a Web page you do so unidentifiably. This is either extremely useful or extremely frustrating, depending upon whether you are the visitor or the visited! My approach was to create a simple HTML [4, 11] form in a Web page. The form allowed a reader of the page to send me their question without revealing their identity. This is possible because the form makes use of a CGI [6] script to deliver the question as an electronic mail message as if it has come from the Web server containing the page. Once the question has been received, and a reply formulated, these are appended to the text of the page containing the form. In this way, everyone on the course can see them including, of course, the original enquirer. The page gradually takes on the character of a mini-FAQ (Frequently Asked Questions) list, reflecting a dynamic dialogue between students and staff. This approach has the added benefit, over the paper-based one, of being available at all times and from anywhere on or off campus where there is Web access.

We have been using Usenet style bulletin boards, in which identity is known, to promote staff-student and student-student since the mid-1980s – Boyle [5] describes a similar use at the University of Leeds. Other authors have used the idea of encouraging questions from students and linking them with Web-based material, but have tended to require students to use straightforward electronic mail in their submission [9], although Arnow [1] stripped mail identification before automatically generating HTML from the text of the mail messages. Hagan [8] does, in fact, use genuine anonymous submission via a similar mechanism to the one described here, but the purpose is to obtain post-assignment feedback. By using the technique within assignments, my approach is to actively encourage the students' analysis and criticism of them, whilst removing any fear of the consequences. Hence, they deepen the learning process associated with those tasks, because their right to ask questions is acknowledged and they receive answers that move them on in their understanding. In software engineering terms, they see refinement of a specification in action, in a context that they can easily relate to. In addition, as a teacher, I receive benefits in making my assignments clearer for future use, and insights into which aspects of the course need elucidating. Even the fact that some questions are asked whose answers seem obvious from the material presented, provides a challenge to find better ways of expressing it.

4 The Form in Action

I have used such forms on several courses in the 1996-97 academic year. The most successful was a course teaching object-oriented design and C++ [13] as a second language to second year Computer Science majors [2]. This involved a two-stage assignment to design and then implement a program to play a simple game of cards. Here is part of the final version of the requirement for the design stage

> Produce a high-level object-oriented design for a program to be written in an object-oriented language to allow the following game of cards to be played amongst 2 or more players.
>
> A game consists of a number of rounds between P players. At the start of each round, every player receives C (52 div P) cards. The remaining (52 mod P) cards are turned face up so that everyone can see them. A single game consists of C rounds in which each player plays a single card in turn. The player of the first card in the first round of a game is decided randomly. Thereafter, the winner of each round plays the first card of the next round of that game, if there is one. After the first card of a round has been played, the remaining players play one card each in turn. These players must follow the suit of the first player if possible. A round is won by the player who plays the highest card of the lead suit. A game is won by the player or players who have won the most rounds in that game.

The students were asked to submit an identification of the classes involved and their interactions - not an implementation.

Here are some of the questions and answers that resulted, showing the approach in practice.

> Q. How does the first player of the round decide what card they should play? Is it the lowest of any suit?
>
> A. This is up to you. Quality of play is not going to be assessed, only adherence to the rules of the game.

I had failed to fully specify the assessment criteria. If more marks were available for better-playing programs, this needed to be known at the design stage.

> Q. How can there be any remaining cards if the number of cards in a pack i.e. 52, is divided by the number of players. Is this for when the number of players doesn't divide evenly into 52?
>
> A. If there are 4 players (P == 4), then each receives 13 cards (C == 52 div 4) with none left over. If there are 5 players, then each receives 10, with 2 (52 mod 5) left over.

A typical question with an 'obvious' answer, but a genuine lack of understanding, nonetheless, and so one that needed addressing.

> Q. Would I be correct in assuming that this game is 'Trumps' without a trump suit?
>
> A. I am not actually sure what this is a version of! I suspect that there are a lot of games with similar rules. It is safest to assume that it is not the same as any game that you already know. In this way you won't be led into making assumptions over issues that have not been properly specified.

An opportunity to make a software engineering point about being careful not to read extra information into a requirement, but to ensure that it is complete in itself.

> Q. Do you use ‘Round’ to refer to both a ‘Trick’ (where every player lays 1 card) and a set of ‘Tricks’ in which all players cards are laid?
>
> A. I have tried to use ‘Round’ as the equivalent of ‘Trick’ only. ‘Game’ is the equivalent of a complete set of C Tricks.

The initial version of the requirement used the words ‘Trick’ and ‘Round’ synonymously. This question led me to remove the potential ambiguity and use ‘Round’ exclusively. This refinement was particularly important because students are often encouraged to identify nouns as potential sources of class definitions.

For the second stage of the assignment, they were required to implement just the ‘Player’ class, given its C++ interface definition. This resulted in several pragmatic questions being asked

> Q. How can I develop the program on my machine at home under DOS during the vacation?

It also produced a number of questions that illustrate the power of this approach

> Q. Would Visual Basic be a good Platform for implementing the design with a ‘Solitaire’ type interface in mind?

This allowed me to highlight the separation of responsibilities between classes, and the opportunities for defining alternative derived classes from a simple ‘Card’ base class if different user interfaces were required.

I had supplied some supporting software with which they could develop their implementations

> Q. Isn’t the manager in main.cc unfair in the order it asks players to play cards?
>
> A. Yes. This was an error on my part and I have now changed main.cc to a fairer version.

It is not always easy to question the correctness of the lecturer’s code. My only regret is that I cannot identify the questioner in order to give them the credit they deserve!

Finally, a genuine question that says a lot for the empowering of this approach

> Q. What does one do if one does not understand any of the C++ you have been talking about in the lectures?

5 Summary

At the heart of our teaching is a desire that our students should *understand* the material that we present to them. We wish to equip them with sufficient understanding that they are then able to develop and *apply* that understanding to the solutions of problems. In order to help them practise these skills, we set them assignments that often mimic real-life problems they may well face in the future. Understanding often requires a dialogue, where questions are raised and answers found. Through the guarantee of anonymity, the method described encourages a wider group of students to ask questions than might otherwise do so. By increasing the approachability of staff to those who would not normally feel able to ask questions directly, we gain valuable insights into the way our students think, and we improve the quality of our course materials and assessment practices.

6 Acknowledgements

I would like to acknowledge the support and encouragement of the Computers and Education Research Group at UKC, in particular Janet Carter, Sally Fincher and Janet Linington who provided valuable input.

References

[1] David M. Arnow and Dayton Clark, 'Extending the conversation: integrating email and web technology in CS programming classes,' *SIGCSE Bulletin*, 28 Special Issue, 1996. pp 93–95.

[2] David Barnes, *CO504 (T204) OO/C++ Assignment – A Card Game*, http://www.cs.ukc.ac.uk/people/staff/djb/t204/cards.html

[3] David Barnes, Sally Fincher and Simon Thompson, *Introductory Problem Solving in Computer Science*, CTI 5th Annual Conference on the Teaching of Computing, Dublin, August 1997.

[4] Tim Berners-Lee and Dan Connolly, 'HyperText Markup Language,' *The World Wide Web Journal*, 1:2, p115–160, 1996.

[5] Roger Boyle, Jim Jackson and Rik Wade, *Changing Learning Culture with Electronic Bulletin Boards*, University of Leeds, School of Computer Studies Research Report 95.2, 1995. ftp://agora.leeds.ac.uk/scs/doc/reports/1995/95_2.ps.Z

[6] *CGI: Common Gateway Interface*, http://www.w3.org/pub/WWW/CGI/.

[7] Tom Gilb, *Principles of Software Engineering Management*, Addison-Wesley, Reading MA, 1988.

[8] Dianne Hagan, *Student feedback via the World Wide Web*, http://ultibase.rmit.edu.au/Articles/hagan1.html

[9] Tomohiro Nishida, Akinori Saitoh, Yoshihiro Tsujino and Nobuki Tokura, 'Lecture Supporting System By Using E-Mail and WWW,' SIGCSE Bulletin, 28:1, March 1996, pp 280–284.

[10] G. Polya, *How To Solve It*, Princeton University Press, 1945.

[11] Dave Raggett, *HTML 3.2 Reference Specification*, World Wide Web Consortium, 1996.

[12] Ian Somerville, *Software Engineering*, Addison-Wesley, Reading MA, 5th ed. 1996.

[13] Bjarne Stroustrup, *The C++ Programming Language*, 3rd ed, Addison-Wesley, Reading, MA, 1997.