A multi-national, multi-institutional study of student-generated software designs

Sally Fincher[1], Marian Petre[2], Josh Tenenberg[3], Ken Blaha[4], Dennis Bouvier[5], Tzu-Yi Chen[6], Donald Chinn[7], Stephen Cooper[8], Anna Eckerdal[9], Hubert Johnson[10], Robert McCartney[11], Alvaro Monge[12], Jan Erik Moström[13], Kris Powers[14], Mark Ratcliffe[15], Anthony Robins[16], Dean Sanders[17], Leslie Schwartzman[18], Beth Simon[19], Carol Stoker[20], Allison Elliott Tew[21], Tammy VanDeGrift[22]

S.A.Fincher@kent.ac.uk

Abstract

This paper reports a multi-national, multi-institutional study to investigate Computer Science students' understanding of software design and software design criteria. Student participants were recruited from two groups: students early in their degree studies and students completing their Bachelor degrees. Computer Science educators were also recruited as a comparison group. The study, including over 300 participants from 21 institutions in 4 countries, aimed to understand characteristics of student-generated software designs, to investigate student recognition of requirement ambiguities, and to elicit students' valuation of key design activities. The results indicate that with experience, students become more aware of ambiguous problem specifications and are able to address more of the requirements in their software designs, that they use fewer textual design notations and more graphical and standardized notations, that they systemically ignore groupings and interactions among the different parts of their designs, and that students change their valuation of key design activities in response to changes in problem-solving context.

1 Introduction

Software design is difficult: dealing with ill-defined and ill-structured problems; having complex and often conflicting constraints; producing large, complex, dynamic, intangible artefacts; and being deeply embedded in a domain, such as finance or medicine (cf. characteristics of the design task described by Goel and Pirolli (1992)). As a result, software design requires a variety of skills and knowledge: within the domain of application, in programming (Soloway and Ehrlich, 1984), and in the mapping between the domain-based problem and software artefacts that carry out the requisite functionality (McCracken, 2004). This paper describes

[1]Computing Laboratory, University of Kent, UK
[2]Department of Mathematics and Computing, Open University, UK
[3]Computing and Software Systems, Institute of Technology, University of Washington, Tacoma, USA
[4]Department of Computer Science, Pacific Lutheran University, USA
[5]Department of Computer Science, Saint Louis University, USA
[6]Department of Mathematics and Computer Science, Pomona College, USA
[7]Computing and Software Systems, Institute of Technology, University of Washington, Tacoma, USA
[8]Department of Mathematics and Computer Science, Saint Joseph's University, USA
[9]Uppsala University, Sweden
[10]Computer Science Department, Montclair State University, USA
[11]Computer Science and Engineering, University of Connecticut, USA
[12]Computer Engineering and Computer Science, California State University Long Beach, USA
[13]Department of Computing Science, UmeåUniversity, Sweden
[14]Department of Computer Science, Tufts University, USA
[15]Department of Computer Science, University of Wales Aberystwyth, UK
[16]Computer Science Department, University of Otago, New Zealand
[17]Department of Computer Science/Information Systems, Northwest Missouri State University, USA
[18]School of Computer Science and Telecommunication, Roosevelt University, USA
[19]Mathematics and Computer Science Department, University of San Diego, USA
[20]Department of Computer Science, Azusa Pacific University, USA
[21]College of Computing, Georgia Institute of Technology, USA
[22]Computer Science and Engineering Department, University of Washington, Seattle, USA

results from a study of the software designs of over 300 Computer Science (CS) students and educators on a simple design task. (A fuller description of these results can be found in (Fincher et al., 2004).)

This study is distinctive from other studies of software design along a number of dimensions. First, it is both multi-institutional and multi-national, with participants from 21 institutions in 4 countries, one of the few software design studies with such a diverse participant pool. Only a multi-institutional study like this allows the assessment of what factors vary across educational contexts - and hence are likely to be influenced by educational intervention - and what are invariant. Second, the data that is examined is particularly rich, with the main components being the written representations and verbal descriptions of participant-generated software designs. This allows many diverse research questions to be addressed using multiple methods of analysis. Third, the study is large-scale, with over 300 participants, which, when combined with the study's multi-institutional nature reduces sample bias and increases generalizability. Given the cost and challenges of carrying out empirical research at this scale, there are few precedents for empirical software studies of this size and scope (but see (McCracken et al., 2001) and (Petre et al., 2003) for other such examples). And fourth, the study includes participants at three different levels of educational attainment, thus allowing the examination of changes in design behaviour with additional formal education.

2 Background

Looking at what Adams et al. (2003) call the *design expertise continuum*, we can gain insight into the different developmental stages of software designers which, it is to be hoped, can be incorporated into more effective design teaching and learning. Jeffries et al. (1981) noted that novices differ from experts in their ability to decompose a software problem effectively, to solve sub-problems, and to integrate solutions. Experts organise information differently from novices, producing different and larger "chunks" (summarised in (Kaplan et al., 1986).) In a study of industrial design engineers, Christiaans and Dorst (1992) found that novices tend to scope out a problem less and seek less information than experienced designers. Rowland (1992) found that novices made few requests for clarifications relative to a design problem.

Expert software practitioners have codified design expertise associated with robust, maintainable, testable, and flexible designs, often focusing on the interaction between different computational modules, as in the design principles of Bruegge and Dutoit (2000): "Ideal subsystem decomposition should minimize coupling and maximize coherence." But even when such principles are taught, it is far from clear that student designers have sufficient skill to apply these principles in practice.

In addition to studying expert/novice differences, some design researchers examine differences in student designers at different stages in their education. For example, Atman et al. (1999) studied differences in the design processes between freshmen and senior engineering students while developing designs for a playground. Not only were there differences in design quality between the two sub-populations, there were differences in design behavior as well. For example, seniors made more requests for information, made more than three times as many assumptions, and made more transitions between design steps, as compared to freshmen. Atman et al. (2003) also examined the design processes of engineering educators so as to provide insight into both educators' actual design practices and its implications for student learning.

In examining student conceptions of design, Newstetter and McCracken surveyed freshmen engineering students by having them rank the five most important and five least important from a list of 16 design activities. They found that the freshmen ranked as least important those activities that are central to general design process descriptions, (e.g. (Goel and Pirolli, 1992)) such as *decomposing*, *generating alternatives*, and *making trade-offs*. Adams et al. (2003), additionally provide evidence that expertise is characterized by matching the design

process to the design context: "experts do not approach every problem in the same way but rather adapt to the inherent constraints of the task."

3 The Study

This study used two tasks to explore students' understanding of the software design process: *a decomposition task*, to examine students' ability to analyse a problem and then design an appropriate solution structure, and to elicit students' understanding-in-action of fundamental software design concepts; and *a design criteria prioritization task*, to elicit which criteria students consider most and least important for different design scenarios.

3.1 Decomposition Task

Participants were given a one-page specification for a "super alarm clock" to help students manage their sleep patterns, and were directed to produce a design meeting these specifications. Participants were asked to "(1) produce an initial solution that someone (not necessarily you) could work from (2) divide your solution into not less than two and not more than ten parts, giving each a name and adding a short description of what it is and what it does - in short, why it is a part. If it is important to your design, you may indicate an order to the parts, or add some additional detail as to how the parts fit together." Participants performed this task individually, without communicating with peers or tutors. On completion, participants were asked to "talk through" their design, and to name and describe the function of each part.

3.2 Design Criteria Prioritization Task

After completing the decomposition task, participants were given 16 cards, each describing a single design criterion. (The phrases represented: Encapsulation, Implementability, High Cohesion, Loose Coupling, Chunking, Intelligibility, Explainability, Parsimony, Re-usability, Recognition of structure, Clarity, Design-phase testing, Maintainability, Engineering, Input re-use, Clear functionality. The phrases as presented can be found in (Fincher et al., 2004)).

Participants were asked to indicate the five most important and the five least important criteria for each of four scenarios. The first scenario was to with respect to the design they had just completed. The participants were then asked to rank the criteria (five most/least important) for each of three hypothetical scenarios:

- for the current task, but in a team (task in team),
- for the current task – on their own – but delivering a fully-functional result at the same time tomorrow (extreme time pressure), and
- for the current task, but designing the system as the basis of a product line that would have a 5-year lifespan (longevity).

3.3 Participants

Participants recruited from 21 institutions of post-secondary education from the USA, UK, Sweden and New Zealand completed the same tasks. Three types of participant were represented from each institution:

First competency students. (FC) To ensure comparability across institutions, students were selected at the point in their education where they could be expected to program at least one problem from the set proposed by McCracken et al. (2001). These problems involve the simulation of a simple calculator for arithmetic expressions. The McCracken problem set was used because it references levels of competence, irrespective of curriculum and was devised for use in one of the first multi-national, multi-institutional

CS Education Research studies. Not all of the FC participants were Computer Science majors, but all had taken, or were taking, a Computer Science course.

Graduating students. (GS) Graduating students were defined to be those within the last eighth of a Bachelor degree program in Computer Science or a related software intensive degree.

Educators. (E) Educators were defined to be those holding faculty positions, and teaching in the undergraduate program.

The total cohort consisted of 314 participants from 21 institutions representing 28 educators, 136 first-competency and 150 graduating students.

For each participant the following material was collected: their representation of the design, the time they took to make it, and a record of their prioritization of the design criteria. Full transcriptions of verbalisation during the task were made for a proportion of the students; researcher notes were made for all.

4 Results and Discussion

Three independent analyses were undertaken to provide different perspectives on the data that was collected. Each analysis is distinguished by the questions explored and the methods used. Exploratory, data-driven analysis of the design artefacts was undertaken to answer questions about the types and characteristics of representations that participants used. A directed qualitative analysis focussed on participants' recognition of ambiguity in the problem specification and in their information-seeking behaviour. And a quantitative analysis was used to answer questions concerning participants' prioritization of the design criteria.

4.1 Characterisation of Design Artefacts

4.1.1 Design Representations

This part of the study was a data-driven examination of the "marks on paper" representations. A sample of designs were first examined in order to develop a set of distinct categories into which each design representation would be grouped. As software practicioner-researchers, we developed these categories to represent semantically meaningful differences in design notation. The categories are:

Standard Graphical: recognised notations of software design, such as Class Diagram, or Entity-Relationship Diagram.

Ad-hoc Graphical: diagrams of any form that were not recognised as standard notations of software design.

Code or pseudo-code: code segments such as assignments, iteration and selection.

Textual: free text descriptions with at most an occasional illustrative diagram.

Each design artefact was visually examined, and categorised into exactly one of the previous disjoint groupings based on its predominating characteristic, or into the category **Mixed** if there was no clear dominance among the other categories.

To ensure consistency the designs were all categorised by three of the authors and assignment to a category required consensus. Figure 1 shows the results of this analysis. The data show a shift from textual to standard graphical representations with increases in education, with the frequency differences between the different subpopulations statistically significant at the $\alpha = .001$ significance level using the χ^2 test. While 47% of FC participants used predominantly textual representations, only 27% of GS participants and 23% of E participants did so. These numbers are inverted for standard graphical representations, with 46% of E participants, 27% of GS participants, and 15% of FC participants using these representations.

4.1.2 Design Complexity

Two indicators of design complexity were examined: the use of grouping structures among parts, and whether the design contained an indication of interaction among the parts. Each researcher analysed the designs from their own institution in terms of grouping by answering the question "Did the design include any hierarchical, nested, or grouping structure of any kind?" For example, a diagram with boxes labelled *Pocket PC*, *Alarm Handler*, and *User Interface*, collectively labelled as *User/Front End* would count as grouping. Similarly indication of interaction was analysed by each researcher answering the question "Are interactions between any of the parts indicated?" For example, a diagram with two boxes, an arrow linking the two boxes, and an explanation that one box is providing information to the other box would count as interaction.

There was some difference in frequency of use of grouping structures between the participant subpopulations; 24% of FC, 27% of GS and 46% of E participants used grouping, with the difference between the combined student groups and the educators significant at the $\alpha = .025$ significance level using the χ^2 test.

There was also significant difference in frequency of use of interaction between the participant subpopulations; 66% of FC, 81% of GS, and 93% of E participants indicated interaction, with the difference between these significant at the $\alpha = .001$ level using the χ^2 test, and significant at the $\alpha = .05$ level when the student groups are combined.

For both complexity measures there are marked institutional differences. For grouping it ranges from a low of 5% of the participants from one institution who included grouping in their designs to a high of 86% of the participants of another institution Q who did so. For those who indicated interaction in their designs, it ranges from a low of 40% of the participants from one institution to a high of 100% of the participants of three institutions who did so. This suggests that there is a strong effect on design depending on how this material is taught.

4.2 Recognising Ambiguity in Requirements

An analysis was conducted to investigate participants' recognition of ambiguous aspects of the design brief requirements. Recognizing and addressing ambiguity is important because it is cheaper to recognize and resolve ambiguities early, rather than after the design is completed (Boehm, 1981).

A participant is called an ambiguity *recognizer* if they ask a question or make an assumption, either in the written representation or verbally during the decomposition task. A participant is an *information gatherers* if they ask questions, whether or not they make observable assumptions. 216 participants are recognizers and 87 are non-recognizers, with 11

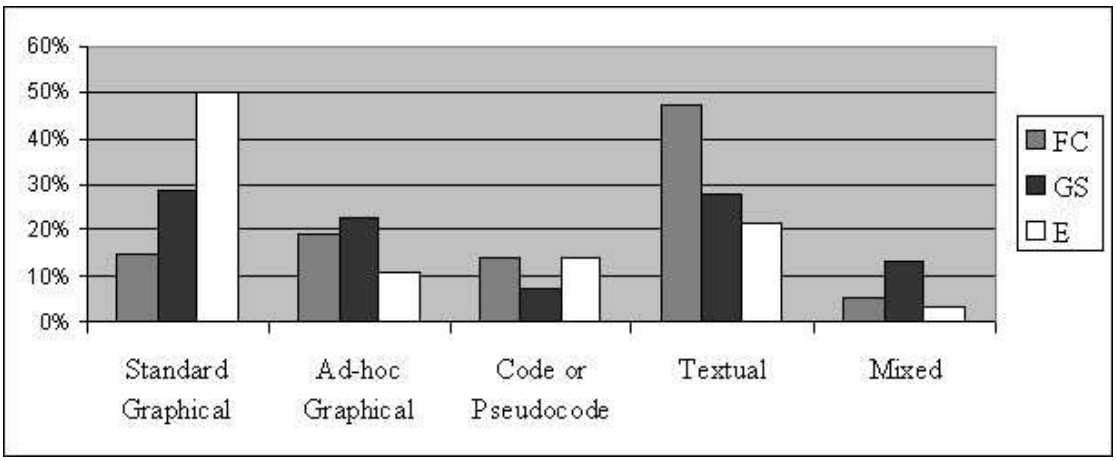

Figure 1: Distribution of Design Representations

participants not being reliably classified. The percentage of recognizers increases with education, with 63% of first competency students being recognizers, 76% of graduating seniors being recognizers, and 89% of educators being recognizers. There are 138 information gatherers and 165 non information gatherers, with no data for 11 participants. As with the recognizers the percentage of information gatherers increases with education; 33% of first competency students, 50% of graduating seniors, and 81% of educators gathered information during the decomposition task.

The information gatherers and recognizers were also analyzed with respect to the number of requirements addressed, with each participant classed into one of four categories: *all* requirements, *most* requirements (>= 50%), *some* requirements (< 50%) *no* requirements. The general trend is that as the number of requirements addressed decreases, the percentage of recognizers also decreases. This indicates that those who recognized ambiguity had a higher success rate in addressing all requirements than those who did not recognize ambiguity. Participants who gathered information also had a higher success rate in addressing all requirements than those who did not gather information. This trend indicates an association between recognizing ambiguity and successfully fulfilling requirements. Once again, institutional differences were evident. In five institutions all participants were recognizers, while in three institutions less than half of the participants were recognizers.

4.3 Design Criteria Prioritization

One of this study's focal questions is whether students recognize different criteria within the design process. This was motivated by discussion with educators and papers such as (CMM Correspondence Group, 1997), as well as textbooks such as (Bruegge and Dutoit, 2000) that suggest that there are particular criteria that should be considered when doing software design. The particular focus here is on the relative importance of different criteria and how these rankings vary over different participant groups and contexts. By examining these prioritizations across participant groups, it could be possible to see how (or whether) these are learned through the curriculum. These rankings were collected immediately after, and are set in the context of, the decomposition task. The data for each prioritization were collected into frequency counts for each participant group and scenario.

Figure 2 shows the number of times each criterion was ranked as one of the five most important criteria by each participant group for the design task that the participants just completed. There is a surprising similarity of valuations across participant groups. Similar such graphs for both "most" and "least" prioritizations in each of the four scenarios can be found in (Fincher et al., 2004); for reasons of space they are not included here.

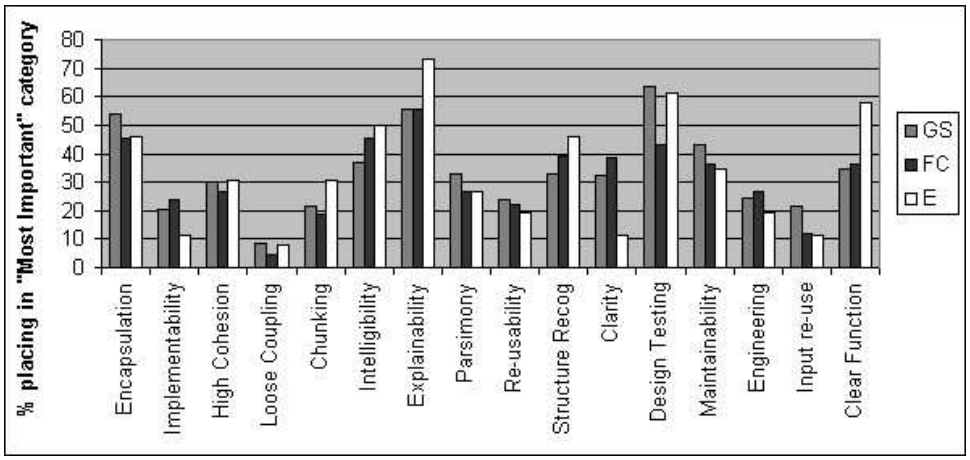

Figure 2: Most important, "current task" scenario

A number of χ^2 tests were run to determine if any apparent differences in frequencies were

statistically significant, across all scenarios for "most" and "least" priorotizations. In terms of between-group agreement, Educator criteria frequency counts were significantly different from those of the students for many of the individual criteria under at least one scenario. However, the frequencies of first competency and graduating students differed little, regardless of scenario. One of the surprising results is that the criteria *Loose Coupling* receives the fewest or second fewest number of counts for most important design criteria by FC students in every scenario. *Loose Coupling* also receives the fewest or second fewest number of counts for most important design criteria by GS students, except in the longevity scenario, where the difference in frequency with FC students for this criterion was the only statistically significant difference between the student groups across all criteria and all scenarios. So although this principle is sacrosanct for practicioners, it is rarely valued as such among students.

One of the things assessed is the degree to which individuals adjust the rankings of their design criteria when faced by different situations. Adams et al. (2003) observed that expert designers adapt the way they approach problems to match task constraints. The results of the present study indicate that students change their prioritizations when faced with different scenarios more than do educators, but there is no statistically significant difference in the amount of this change between the two student groups.

5 Conclusion

Each of the three analyses yielded results, with the main ones summarized here.

Design characteristics: there is a progression away from the textual and toward standard graphical notations with increases in education. The data also indicate that though a large number of students underestimate the importance of representing structural groupings and interactions between design parts, differences in institutional characteristics, including how software design is taught, might account for these differences.

Recognition of ambiguity: the percentage of both information gatherers and recognizers of ambiguity increases from first competency students to graduating students to educators. And those who recognize ambiguity or gather information had a higher success rate in addressing all requirements than those who did not. As with representation characteristics, there was considerable institutional difference in frequency of ambiguity recognizers and information gatherers.

Design criteria: there was little indication of changes to valuation of design criteria among students with differences in education. Each participant group changed its prioritizations in response to changing design contexts; surprisingly, students were more flexible, and adapted their criteria rankings to the context of the task to a greater degree than did educators.

Taken in total, these results suggest the following. First, that some design behaviors appear to be *developmental*, such as recognition of ambiguity and use of standardized design representations, in that there are increases in the occurrence of these behaviors with increases in educational attainment. Second, some design behaviors appear relatively invariant with respect to different levels of education within the Bachelor degree, such as design criteria valuation. It is possible that changes to these behaviors, such as appreciation of certain design criteria, is obtained primarily as a result of hard-won experience in "real-world" software development contexts. And third, some design behaviors are context-dependent, such as information gathering and representation of part-part interactions, suggesting that these behaviors are most amenable to changes in instruction.

6 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. DUE-0243242. Any opinions, findings, and conclusions or recommendations expressed in

this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Thanks to Leigh Waguespack for assistance with statistical analysis and to Janet Rountree for assistance with data gathering and transcription.

References

Adams, R. S., Turns, J., Atman, C. J., 2003. What could design learning look like? In: Expertise in Design: Design Thinking Research Symposium 6. Sydney, Australia.

Atman, C. J., Chimka, J. R., Bursic, K. M., Nachtmann, H. L., 1999. A comparison of freshman and senior engineering design processes. Design Studies 20, 131–152.

Atman, C. J., Turns, J., Cardella, M., Adams, R. S., 2003. The design processes of engineering educators: Thick descriptions and potential implications. In: Expertise in Design: Design Thinking Research Symposium 6. Sydney, Australia.

Boehm, B., 1981. Software Engineering Economics. Prentice Hall.

Bruegge, B., Dutoit, A., 2000. Object-Oriented Software Engineering. Prentice Hall.

Christiaans, H. H. C., Dorst, K. H., 1992. Cognitive models in industrial design engineering. Design Theory and Methodology 42 (131-140).

CMM Correspondence Group, October 1997. Software product engineering (draft). Technical Report.

Fincher, S., Petre, M., Tenenberg, J., et al, September 2004. Cause for alarm?: A multi-national, multi-institutional study of student-generated software designs. Tech. Rep. 16-04, Computing Laboratory, University of Kent, Canterbury.
URL `http://www.cs.kent.ac.uk/pubs/2004/1953`

Goel, V., Pirolli, P., 1992. The structure of design problem spaces. Cognitive Science 16, 395–492.

Jeffries, R., Turner, A. A., Polson, P. G., Atwood, M. E., 1981. The processes involved in designing software. In: Anderson, J. (Ed.), Cognitive Skills and their Acquisition. Lawrence Erlbaum Associates.

Kaplan, S., Gruppen, L., Levanthal, L. M., Board, F., 1986. The components of expertise: a cross-disciplinary review. Tech. rep., University of Michigan.

McCracken, W. M., 2004. Research on learning to design software. In: Fincher, S., Petre, M. (Eds.), Computer Science Education Research. Routledge Falmer, Lisse, pp. 155–174.

McCracken, W. M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., Wilusz, T., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. SIGCSE Bulletin 33 (4), 125–180.

Petre, M., Fincher, S., Tenenberg, J., et al, June 2003. "My Criterion is: Is it a Boolean?": A card-sort elicitation of students' knowledge of programming constructs. Tech. Rep. 6-03, Computing Laboratory, University of Kent, Canterbury, Kent, UK.
URL `http://www.cs.kent.ac.uk/pubs/2003/1682`

Rowland, G., 1992. What do instructional designers actually do? Performance Improvement Quarterly 5 (2), 65–86.

Soloway, E., Ehrlich, K., 1984. Empirical studies of programming knowledge. IEEE Transactions on Software Engineering 10 (5), 595–609.