Students designing software: a multi-national, multi-institutional study

Josh Tenenberg[1], Sally Fincher[2], Ken Blaha[3], Dennis Bouvier[4], Tzu-Yi Chen[5], Donald Chinn[6], Stephen Cooper[7], Anna Eckerdal[8], Hubert Johnson[9], Robert McCartney[10], Alvaro Monge[11], Jan Erik Moström[12], Marian Petre[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]

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 criteria. The results indicate that with increases in education, students use fewer textual design notations and more graphical and standardized notations and that they become more aware of ambiguous problem specifications. Yet increased educational attainment has little effect on students' valuation of key design characteristics.

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;

[1]Computing and Software Systems, Institute of Technology, University of Washington, Tacoma, USA, jtenenbg@u.washington.edu
[2]Computing Laboratory, University of Kent, UK
[3]Department of Computer Science, Pacific Lutheran University, USA
[4]Department of Computer Science, Saint Louis University, USA
[5]Department of Mathematics and Computer Science, Pomona College, USA
[6]Computing and Software Systems, Institute of Technology, University of Washington, Tacoma, USA
[7]Department of Mathematics and Computer Science, Saint Joseph's University, USA
[8]Uppsala University, Sweden
[9]Computer Science Department, Montclair State University, USA
[10]Computer Science and Engineering, University of Connecticut, USA
[11]Computer Engineering and Computer Science, California State University Long Beach, USA
[12]Department of Computing Science, Umeå University, Sweden
[13]Department of Mathematics and Computing, Open University, UK
[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

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). Even in professional design behaviour, there are a number of constituent and interacting skills – and potential sources of breakdown – that include such things as marshalling resources, applying knowledge, prioritising sub-tasks, managing constraints, evaluating proposed solutions, and managing the design process (Guindon et al., 1987; Curtis, 1990). These characteristics make software design elusive to characterize and difficult to teach.

This paper describes results from a study of the software designs of over 300 Computer Science (CS) students and educators given a simple design task. (A more complete 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 factors 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

Models of design in general and software design in particular involve decomposition and management of the design process (Détienne, 2001; Goel and Pirolli, 1992). This management includes tracking the relationships among sub-problems and integrating sub-problems into a coherent structure. 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 organize information differently than novices, producing different and larger "chunks" (summarized 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. Bogush et al. (2000) found

that freshmen engineers tend to define problems narrowly while more experienced seniors tend to define problems more broadly. And Atman et al. (1999) found differences in both design quality and design behavior between freshmen and senior engineering students. 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 (2001) 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." In order to reduce the likelihood of introducing bias, participants were prompted with generic language (e.g., *part* rather than *object* or *function*), to elicit their concept of what constitutes a part, and how to describe and represent parts. 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. The verbatim design brief given to participants is provided in Appendix A.

3.2 Design Criteria Prioritization Task

After completing the decomposition task, participants were given 16 cards, each with a phrase describing a single design criterion. The phrases represented: Encapsulation, Implementability, High Cohesion, Loose Coupling, Chunking, Intelligibility, Explainability, Parsimony, Reusability, Recognition of structure, Clarity, Design-phase testing, Maintainability, Engineering, Input re-use, and Clear functionality. The phrases can be found in Appendix B.

Participants were asked to indicate the five most important and the five least important criteria for each of four scenarios:

- for the design they had just completed (*current task*),
- 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 within the study population:

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 undergraduate Computer Science (or related) programs.

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 written representation of the design (their "marks on paper"), the number of parts in their design and their name for each part, the time they took to make the design, and a record of their prioritization of the design criteria. Full transcriptions of verbalisation during the task were made for a subset 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. These categories were developed to represent semantically meaningful differences in design notation to the practitioner-researchers undertaking this research. The categories are:

Standard Graphical: This was used to include recognized notations of software design. Ten different types were represented in the corpus: Architecture Diagram, Class Diagram, Class-Responsibility-Collaborator (CRC) Cards, Data Flow Diagram (DFD), Entity-Relationship Diagram (ER), Flowchart, Graphical User Interface (GUI), Sequence Diagram, State Transition Diagram (STD) and Use Case Diagram.

Ad-hoc Graphical: This category included diagrams of any form not recognized as standard notations of software design. Large sections of text were accepted in this category providing that they were considered refinements of items identified in the diagram. In some cases it was difficult to differentiate between *ad hoc* diagrams and standardized graphical representations. In order to characterize the latter category, detailed syntax was ignored and benchmarks defined. For example in order to be recognized as a Class Diagram, it was agreed that a representation should consist of a box conceptualising both data and functions.

Code or pseudo-code: This was used for any software design that included code segments such as assignments, iteration and selection.

Textual: This category was used for free text descriptions but allowed an occasional diagram used for illustration: for example, graphical interface or report layout.

Mixed: This was used when there was no clear dominance between different styles. For example a participant might start with a textual description then proceed with a Class Diagram. If there was no connection between the descriptions and the identified classes then the category was Mixed (Text and Class Diagrams).

Examples of the different representation categories are provided in Figure 1.

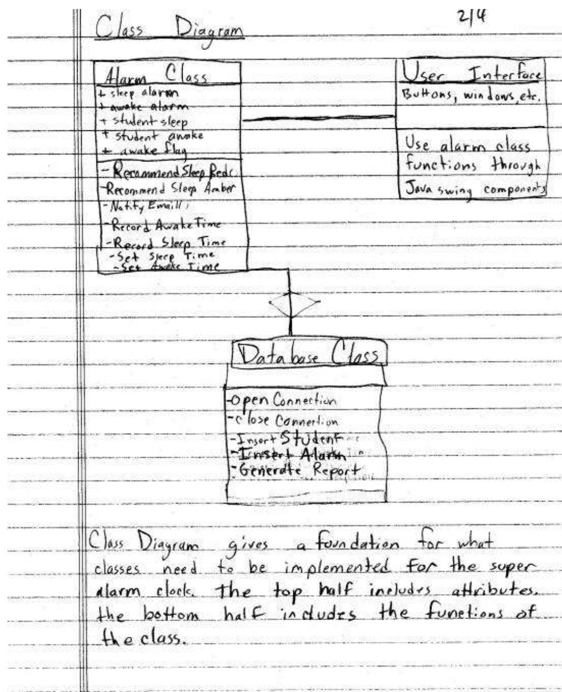
Each design artefact was visually examined, and categorized into exactly one of the previous disjoint groupings based on its predominating characteristic. To ensure consistency the designs were categorized by three of the researchers and assignment to a category required consensus. Figure 2 shows the 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 28% of GS participants and 21% of E participants did so. These numbers are the opposite for standard graphical representations, with 50% of E participants, 29% of GS participants, and 15% of FC participants predominantly using standard graphical representations.

4.1.2 Design Complexity

Three indicators of design complexity were examined: 1) the number of parts in each design, 2) the use of grouping structures among parts, and 3) whether the design contained an indication of interaction among the parts. This analysis excluded data from one first-competency subject whose design representation was uninterpretable by the researchers.

Number of Parts. The design task protocol was specified so as to capture the number of parts for each design, and the name of each part. Table 1 provides descriptive statistics for each participant subpopulation. A one-way ANOVA shows that the variation in the mean number of parts for FC (5.1), GS (4.6) and E (6.2) participant groups is highly significant ($p = 0.001$), although there is no clear trend over the educational level of these subgroups (note that GS has the lowest mean). Variation in mean number of parts by institution (data not shown) is also significant ($p < 0.01$).

Grouping Structures. 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. There was some difference in frequency of use of grouping



(a) Standard Graphical

1/4

Part 2: Bookkeeping

This is easy. The program simply needs to, when told, write down on the hard drive in some bookkeeping file "about to sleep" xor ("woken up" plus the time of the last alarm) depending upon which it is told.

Part 3: Nagging

Periodically, the program needs to check the above mentioned file to find out when

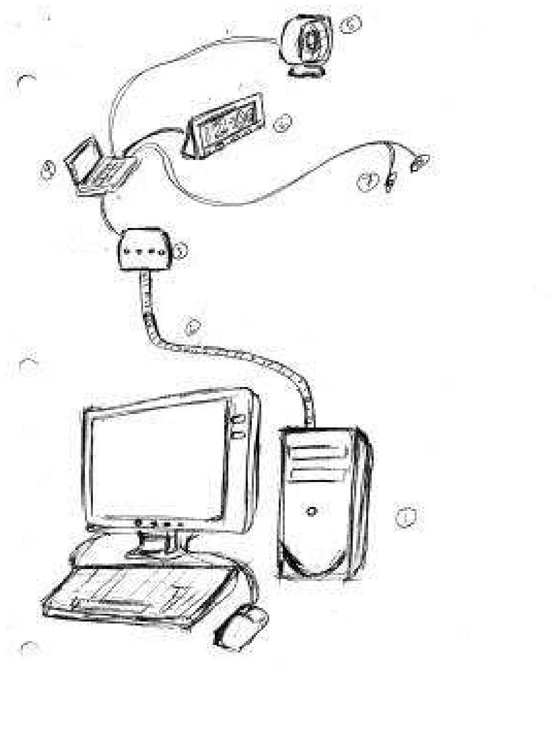
The program should remember when it is told "about to sleep" and issue an "amber alert" or "red alert" as appropriate (consult a sleep specialist to find out when).

There is an expected server side mate to this program which will issue orders (via TCP/IP presumably) to the program.

Communication: Encryption is important for privacy reasons. Use a secure channel (i.e. secure socket layer, or SSL).

When told by the server, the program needs to issue

(b) Textual



(c) Ad-hoc Graphical

```
Student Interaction                                              3/4
// most likely a module used for interaction with the student
// used like a GUI, has input devices (could be buttons) to
// recieve input from the student

 input devices:
    setalarmwake, setalarmsleep, goingtosleep, wokenupAla
    wokenupOther; reportRequest
 output devices:
    recomendations, reports

 if (set alarm wake)
    request alarm time from student
    Alarm.setwake (wake time)
    write to DB (setwake, wake time)
 else if (setalarmsleep)
    request alarm time from student
    Alarm.setsleep (sleep time)
    write to DB (setsleep, sleep time)
 else if (wokenup Alarm)
    if (Alarm.getWake() + 2 ≤ current time)
       write to DB (wokenup Alarm)
 else if (wokenup Other)
    if (Alarm.getWake() + 2 ≤ current time)
       write to DB (wokenup Other)
 else if (report Request)
    display (get from DB (report))
```

(d) Code

Figure 1: Sample Design Representations

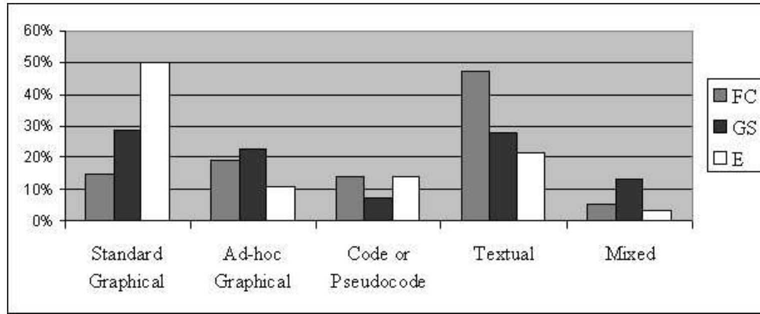


Figure 2: Sample Standard Graphical Representation

	N	Min	Max	Mean	Median	Mode	SD
FC	135	2	11	5.1	5.0	3.0	2.2
GS	150	2	12	4.6	4.0	4.0	1.9
E	28	3	14	6.2	5.5	4.0	2.9

Table 1: Descriptive statistics for number of parts

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, but with no significant difference between the student groups.

There were, however, marked differences in participant responses based on institution, ranging from a low of 5% of the participants from one institution who used grouping structures to a high of 86% of the participants of another institution who did so.

Interactions among Parts. Each researcher also analysed the designs from their own institution in terms of interaction by 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 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.

On this measure there was also marked differences in participant responses based on institution, ranging from a low of 40% of the participants from one institution who indicated part interactions to a high of 100% of the participants of three institutions who did so.

Examining the actual design artefacts that students produce provides evidence that their formal education moves students along the design expertise continuum (Adams et al., 2003). With increases in education, participants used increasing amounts of notations that are standards among practicing software developers, relying less on the predominantly textual descriptions.

And along with increases in educational attainment there were also corresponding increases in the inclusion of part-part interaction and groupings within the designs, but to a less extent than that exhibited by the Educators. This provides evidence that in moving from novices to graduates, students are developing the skills for decomposing problems into sub-problems, understanding the relationships among sub-problems, and composing the parts that solve

sub-problems into a coherent structure, essential characteristics of all design (Détienne, 2001; Goel and Pirolli, 1992).

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 ambiguities in requirements can propagate to errors in the design solution. It is cheaper to recognize and resolve ambiguities early, rather than after the design is completed (Boehm, 1981). Thus, recognizing ambiguity in the design phase is less costly in terms of time to completion and number of bugs. Observed differences between participant groups with respect to recognizing and resolving ambiguity can provide insight into ways to enhance the education of future software designers. For example, if ambiguity is not commonly part of homework assignment specifications, then students may not have practice in recognizing it.

To study the question regarding participants' recognition of ambiguity and the level to which they address requirements the following questions are asked of the corpus:

1. Did the participant ask questions about ambiguities and omissions in the specification (as distinct from questions about word meanings or procedural questions)? (Yes or No)

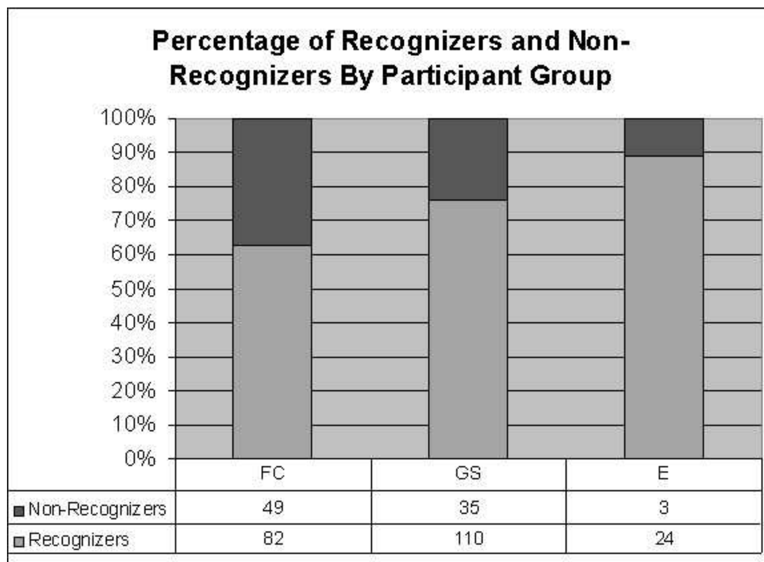
2. Did the participant make explicit assumptions in the description, representation or other recorded responses about ambiguities and/or omissions in the specification? (Yes or No)

3. Did the subject attempt to address the requirements of the specification? (Yes: 100% of requirements addressed, Partially: $\geq 50\%$, Hardly: $< 50\%$, No: 0%)

The data for the separate analyses excludes participants for which the question associated with the analysis could not be answered based on observable behavior and written representation.

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. The *information gatherers* comprise that subset of the recognizers who ask questions.

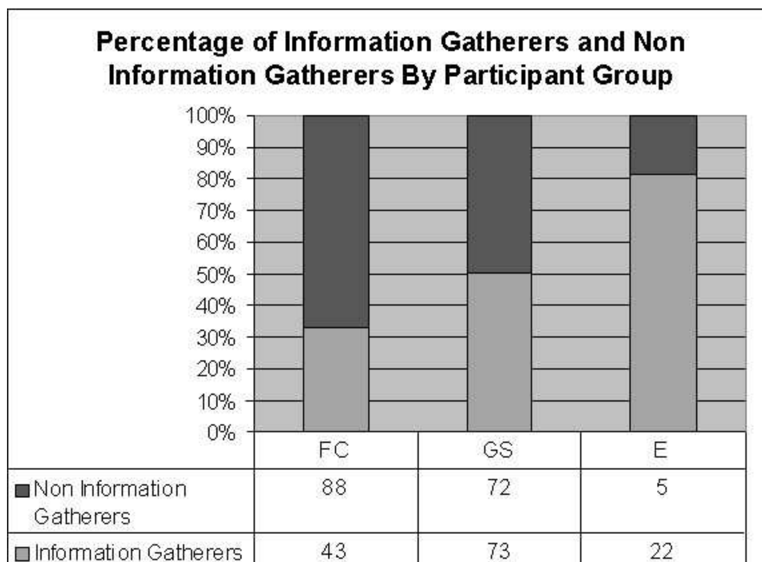
Figure 3 indicates the percentage of participants who recognized ambiguity, by participant group. 216 participants are ambiguity recognizers and 87 are non-recognizers, with 11 participants not being reliably classified. The percentage of recognizers increases with education: 63% of first competency students, 76% of graduating seniors, and 89% of educators.



	FC	GS	E
Non-Recognizers	49	35	3
Recognizers	82	110	24

Figure 3: Ambiguity Recognizers by Participant Group

Figure 4 indicates the percentage of participants who asked questions, by participant group. 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.



	FC	GS	E
Non Information Gatherers	88	72	5
Information Gatherers	43	73	22

Figure 4: Information Gatherers by Participant Group

Figures 5 and 6 show for each requirement-addressing categories (*yes, partially, hardly, no*), the percentage of recognizers and information gatherers. The general trend is that as the number of requirements addressed decreases, the percentage of recognizers and information gatherers also decreases. This indicates that those who recognized ambiguity and gathered information had a higher success rate in addressing all requirements than those who did not. As was the case with the examination of design characteristics, institutional differences were evident. In five institutions all participants were recognizers, while in three institutions less than half of the participants were recognizers.

The percentage of recognizers and information gatherers increases from first competency students to graduating seniors to educators. Information gatherers are a subset of recognizers, so this relationship is not surprising. Proportionally more seniors recognize ambiguity and seek information than first competency students, consistent with the results reported in (Atman et al., 1999), who observed that more seniors request information than freshmen. While only half of those in the first competency group who made assumptions requested additional information, over 90% of educators recognizing ambiguity requested additional information.

The results from analyzing participants' information-gathering, assumption-making, and requirements-addressing indicate that as students go from first-competency to graduating seniors, they tend to recognize ambiguities in under-specified problems. Additionally, participants who recognized ambiguity (and the subset who gathered information) had a higher success rate in addressing all requirements. These results imply that with experience, students will become more aware of ambiguous specifications and by realizing that ambiguities exist, they can design software that meets requirements. This suggests that educators should be more explicit in teaching students how to recognize ambiguity in problem specifications.

4.3 Design Criteria Prioritization

As described in Section 3.2, after completing the decomposition task, participants were asked to indicate the five most and five least important design criteria for each of four scenarios.

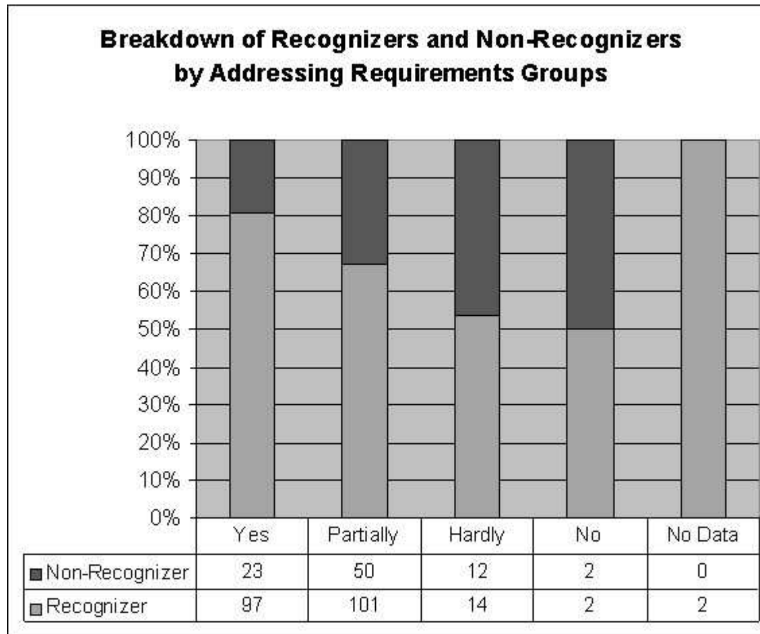


Figure 5: Percentage of Recognizers by Requirement-Addressing Group

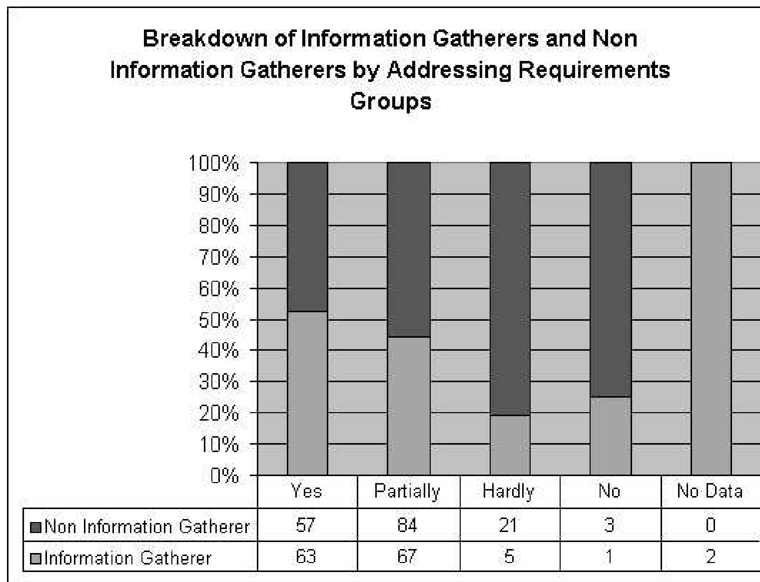
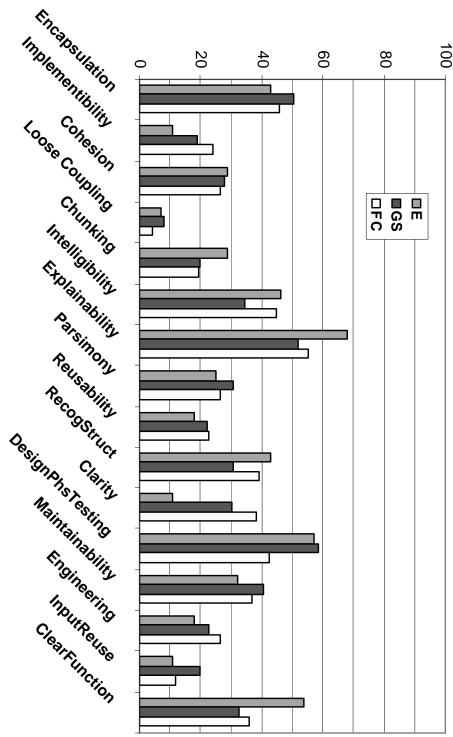


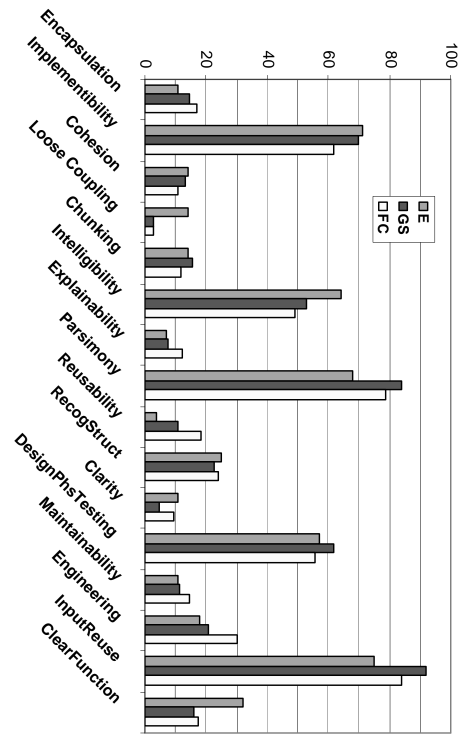
Figure 6: Percentage of Information Gatherers by Requirement-Addressing Group

Their selections were collected into frequency count tables which were then statistically analyzed in order to better understand how priorities vary over different participant groups and design scenarios. This task was motivated by discussion with educators and examination of the practitioner literature such as (CMM Correspondence Group, 1997), (Bourque et al., 2002), and (Bruegge and Dutoit, 2000) that suggest there are particular criteria that should be considered when doing software design. By examining these prioritizations across participant groups, it could be possible to see how (or whether) these are learned through the curriculum.

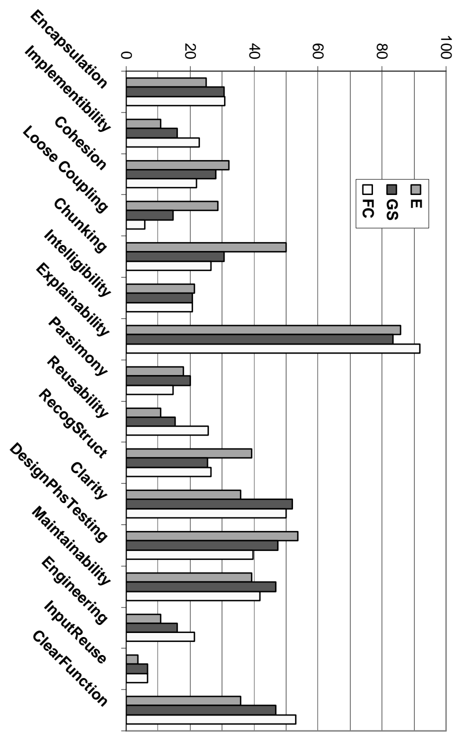
Figure 7 shows the number of times each criterion was ranked as one of the five most important criteria by each participant group in each of the four scenarios.



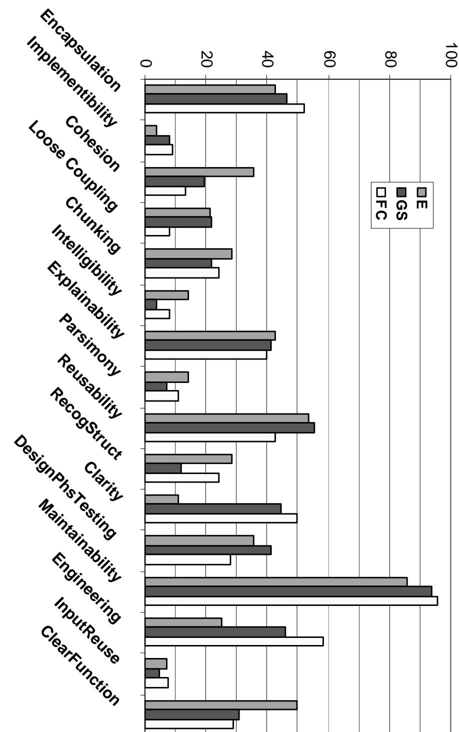
(a) Current task



(b) Time pressure



(c) Task in Team



(d) Longevity

Figure 7: The criteria chosen as most important by participant group and scenario. The numbered axis gives the percent of subjects who classified that criterion as one of the five most important.

A few observations are worth noting. First, there appears to be considerable agreement across the groups as to what criteria are important in each scenario, although the agreement is most pronounced under the time pressure scenario. And second, some of the criteria are ranked as relatively important in most scenarios (e.g. *design-phase testing*), some have importance that is highly scenario-dependent (e.g. *maintainability*), and some are never ranked as important in any of the four scenarios (e.g. *loose coupling*).

4.4 Agreement between subpopulations

In order to determine if any of the apparent differences in criteria valuation between participant groups is statistically significant, χ^2 tests were performed on 3×16 contingency tables such as those in Table 2. The rows represent participant groups, the columns represent the individual criteria, and a cell (i, j) represents the number of participants from participant group i ranking criteria j as most important.

Scenario X

	Criterion$_1$	Criterion$_2$	...	Criterion$_{16}$
FC	$O_{FC,1}$	$O_{FC,2}$	...	$O_{FC,16}$
GS	$O_{GS,1}$	$O_{GS,2}$	...	$O_{GS,16}$
E	$O_{E,1}$	$O_{E,2}$	...	$O_{E,16}$

Table 2: Arrangement of data in 3×16 contingency table. $O_{i,j}$ is the number of participants from group i ranking Criterion$_j$ most important in Scenario X.

There were four such contingency tables, one per scenario, giving four null hypotheses of the form "there is no statistically significant difference between the three subpopulations in their choices for the most significant criteria in this scenario."

At the $\alpha = 0.05$ level, the criteria chosen by different groups differed significantly only for the Longevity scenario; *cohesion* was chosen more by E than the other two subpopulations, *clarity* was chosen less by E, and *loose coupling* was chosen less by FC.

An additional observation is that four of the criteria were never considered among the five most important by either student group in any scenario: *loose coupling*, *cohesion*, *chunking*, and *recognition of structure*. In fact, *loose coupling* is one of the two least frequently chosen criteria by FC students in every scenario. *loose coupling* is also one of the two least frequently chosen design criteria by GS students in each scenario except in the longevity scenario. Note, however, that neither *loose coupling* nor *cohesion* are ever among the five most important criteria for educators as well. So although this principle is sacrosanct for practitioners, it is rarely valued as such among students or their educators, especially when compared to criteria that might be considered more pragmatic, such as those related to design expression (e.g. *explainability*) or process (e.g. *design-phase testing*).

4.5 Variations across scenarios

As seen in Figure 7, the prioritization can change dramatically across scenarios – the differences between these scenarios are much larger than the differences between groups within either scenario. The final set of tests consider the degree to which any individual participant adjusts his or her rankings of the design criteria as the scenarios change. The measure of how much an individual's response varied by scenario was calculated as follows:

1. Each criteria ranking for a single scenario was expressed as a 16-element vector, with positions corresponding to criteria. Each vector element was given the value -1 if the criterion was in the least important partition, 1 if it was in most, and 0 otherwise.
2. The mean of the four scenario vectors was calculated.

3. The squared Euclidian distances between the mean vector and each of the four scenario vectors were computed and then summed.

This sum was taken as a measure of individual variation in criteria priority rankings. An individual giving the same rankings to criteria in each scenario would have a sum of 0, while an individual giving maximally different rankings across scenarios would have a sum of 40. The average sums for the three groups are FC = 23.3, GS = 23.5, and E = 17.9. A one-way analysis of variance indicated a significant difference at the $\alpha = 0.05$ level between the educators and both student groups, with no statistically significant difference between the student groups. It is thus clear that all participant groups are exhibiting context-specific preferences for many of the design criteria, with students doing so more than educators.

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. In addition, the data indicate that a large number of students underestimate the importance of representing structural groupings and interactions between design parts. However, this might be accounted for by differences, including how software design is taught, between institutions.

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 almost complete uniformity of valuation of design criteria among the different participant groups. Across all scenarios, educators gave statistically significant differences in criteria valuations than students for only two criteria, and students differed from one another for only one criteria in a single scenario. Principles of loose coupling and high cohesion were not valued highly by any participant group in any scenario when compared to such pragmatic considerations as clarity of design expression and management of the design process.

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 interactions between parts, 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. The design brief was developed from a classroom exercise of B.J. Fogg.

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.

Bogush, L. L., Turns, J., Atman, C. J., 2000. Engineering design factors: how broadly do students define problems? In: ASEE/IEEE Frontiers in Education. Kansas City, p. Session S3A.

Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L., Wolff, S., 2002. Fundamental principles of software engineering – a journey. The Journal of Systems and Software 62, 59–70.

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.

Curtis, B., 1990. Empirical studies of the software design process. In: Human-Computer Interaction - INTERACT '90. pp. xxxv–xi.

Détienne, F., 2001. Software Design - Cognitive Aspects. Practitioner Series. Springer Verlag.

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.

Guindon, R., Krasner, H., Curtis, B., 1987. Breakdowns and processes during the early activities of software design by professionals. In: Olson, G. M., Sheppard, S., Soloway, E. (Eds.), Empirical Studies of Programmers: Second Workshop. Ablex, pp. 65–82.

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., et al, 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. SIGCSE Bulletin 33 (4), 125–180.

Newstetter, W. C., McCracken, W. M., 2001. Novice conceptions of design: Implications for the design of learning environments. In: Eastman, C. M., McCracken, W. M., Newstetter, W. C. (Eds.), Design Knowing and Learning: Cognition in Design Education. Elsevier, Amsterdam.

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.

A Design Brief

Getting People to Sleep

In some circles sleep deprivation has become a status symbol. Statements like "I pulled another all-nighter" and "I've slept only three hours in the last two days" are shared with pride, as listeners nod in admiration. Although celebrating self-deprivation has historical roots and is not likely to go away soon, it's troubling when an educated culture rewards people for hurting themselves, and that includes missing sleep.

As Stanford sleep experts have stated, sleep deprivation is one of the leading health problems in the modern world. People with high levels of sleep debt get sick more often, have more difficulties in personal relationships, and are less productive and creative. The negative effects of sleep debt go on and on. In short, when you have too much sleep debt, you simply can't enjoy life fully.

Your brief is to design a "super alarm clock" for University students to help them to manage their own sleep patterns, and also to provide data to support a research project into the extent of the problem in this community. You may assume that, for the prototype, each student will have a Pocket PC (or similar device) which is permanently connected to a network.

Your system will need to:

- Allow a student to set an alarm to wake themselves up.
- Allow a student to set an alarm to remind themselves to go to sleep.
- Record when a student tells the system that they are about to go to sleep.
- Record when a student tells the system that they have woken up, and whether it is due to an alarm or not (within 2 minutes of an alarm going off).
- Make recommendations as to when a student needs to go to sleep. This should include "yellow alerts" when the student will need sleep soon, and "red alerts" when they need to sleep now.
- Store the collected data in a server or database for later analysis by researchers. The server/database system (which will also trigger the yellow/red alerts) will be designed and implemented by another team. You should, however, indicate in your design the behaviour you expect from the back-end system.
- Report students who are becoming dangerously sleep-deprived to someone who cares about them (their mother?). This is indicated by a student being given three "red alerts" in a row.
- Provide reports to a student showing their sleep patterns over time, allowing them to see how often they have ignored alarms, and to identify clusters of dangerous, or beneficial, sleep behaviour.

In doing this you should (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 important to your design, you may indicate an order to the parts, or add some additional detail as to how the parts fit together.

B Design Criteria

These are the phrases that were on the 16 cards given to participants. The one or two word "short form" used in the paper (italicized here) was not included on the cards.

1. *[encapsulation]* Hiding the internal workings of each part of the solution from the user, presenting them with a simple interface to its functionality.
2. *[implementability]* Knowing how each part of the solution could be implemented.
3. *[cohesion]* Making sure related things appear together.
4. *[loose coupling]* Making sure that un-related things are linked via a narrow (internal) interface.
5. *[chunking]* Making sure the design is made up of appropriately-sized "chunks".
6. *[intelligibility]* Being able to explain what each part of the solution is, and what it does, to yourself.
7. *[explainability]* Being able to explain what each part of the solution is, and what it does, to others.
8. *[parsimony]* Constructing a solution using the simplest thing that gets the job done.
9. *[re-usability]* Working to achieve a solution of maximum generality.
10. *[recognition of structure]* Ensuring that the parts which make up the solution map onto the structure of the problem.
11. *[clarity]* Designing so that someone else can implement the solution with little (or no) additional information or domain expertise.
12. *[design-phase testing]* "Sanity-checking" the solution, by checking back to the specification.
13. *[maintainability]* Designing a system that can be easily maintained.
14. *[engineering]* Considering the technological implementation (target platform or device) and designing for efficient use of that resource.
15. *[input re-use]* Using ideas that I know work.
16. *[clear functionality]* Expressing the functionality clearly.