Do Students Recognize Ambiguity in Software Specifications?
A Multi-national, Multi-institutional Report

Tammy VanDeGrift¹, Beth Simon², Dean Sanders³, and Ken Blaha⁴

¹ Electrical Engineering & Computer Science, University of Portland, Portland, OR 97203
vandegri2@up.edu
² Computer Science & Engineering, University of California, San Diego, La Jolla, CA 92093-0404
esimon@cs.ucsd.edu
³ Computer Science & Information Systems, Northwest Missouri State University, Maryville, MO 64468
sanderson@nmissouri.edu
⁴ Computer Science & Computer Engineering, Pacific Lutheran University, Tacoma, WA 98447
blahakd@plu.edu

Abstract. Successful software engineering requires experience and acknowledgment of complexity, including that which leads designers to recognize ambiguity within the software design description itself. We report on a study of 21 post-secondary institutions from the USA, UK, Sweden, and New Zealand. First competency and graduating students as well as educators were asked to perform a software design task. We found that graduating seniors were more likely to recognize ambiguities in under-specified problems than first competency students. Additionally, participants who addressed all requirements in the design were more likely than others to recognize ambiguities in the design specification. The behavior of recognizing ambiguity and gathering information appear to be independent of past performance, as measured by course grades.

1 Introduction

Software engineering requires many skills, some of which include gathering information about the domain, designing, implementing, testing, debugging, and documenting [13,12]. Tension between prioritization of sub-tasks, evaluation of proposed solutions, and the constant management of process can lead to breakdowns in the design process, even among professionals [9]. Among the many skills involved in design is the ability to recognize ambiguity in a software engineering project. Recognizing ambiguity is an important part of the design process in that it enables designers to gather information and refine assumptions. The book Are Your Lights On? suggests that a designer should raise several questions when trying to understand a problem's specification [8]. Hilburn states
that students need more and better training for the inspection of formal specifications [10].

This work reports results from a multi-national, multi-institutional study of student-generated software designs [6]. We describe results pertaining to participants' recognition of ambiguity in the design specification used to prompt initial software designs. We say that a participant recognized ambiguity by asking questions, other than those of process, or by making assumptions during the design process.

Prior work in comparing the design processes of freshman and senior engineering students found that seniors made more requests for additional information and made more than three times as many assumptions [5]. Recognizing and addressing ambiguity is important because ambiguities in requirements can propagate to errors in the designed solution. It is cheaper to recognize and resolve ambiguities early in the design process [3]. Recognition of ambiguity in the software design specification reflects design maturity, and our work shows that, across a diverse and multi-national sample, recognition of ambiguity is associated with designs that address more requirements. This work should spur discussion on how to educate students with regard to recognizing ambiguity and its importance in the software design process. If explicitly addressing ambiguity is not currently part of homework assignment specifications, students have little practice in developing this important skill.

Following the introduction, we provide information about the research study, including the subject population, research questions, and the data gathered for analysis. We describe the results from the data collected and offer conclusions based on our work.

2 Research Study

This study reports on 21 post-secondary institutions in the USA, UK, Sweden, and New Zealand participating in the Scaffolding Research in Computer Science Education, an NSF-sponsored workshop [1,6,7].

2.1 Tasks

Participants in the research study were asked to perform two tasks related to software engineering. The first task, called the decomposition task, asked participants to design a solution from a design brief (See Figure 1). Each participant was asked to provide a solution (a design) for a "super alarm clock".

Participants were explicitly invited to ask questions and to take as long as they wished. Upon completion of the design solution, participants were asked to talk about their designs and to describe each part and its function. Their verbal descriptions were recorded and transcribed for later analysis.

Following the decomposition task, participants were asked to perform a design criteria prioritisation task. Participants were given 16 cards, each describing a single design criterion. For example, "Knowing how each part of the solution could be implemented" and "Making sure that un-related things are linked via a narrow (internal) interface" were two of the design criteria. For a complete description of the design criteria, see [6].

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 gets 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.

Fig. 1. Design Brief for the Decomposition Task
2.2 Participants

Three populations were asked to perform the tasks above. The first population, defined as first competency students (FC), are students who could program a simple calculator as defined by the problem in [12]. Not all FC participants were majors in Computer Science, but all had taken the required courses necessary to program a simple calculator.

The second population consisted of graduating students (GS), defined as Computer Science students within the last one-eighth of a Bachelors degree program. Many graduating seniors were completing the final term prior to graduation.

The final population included in the study were educators (E). Educators were defined by those holding faculty positions and teaching Computer Science in the undergraduate curriculum. Details pertaining to the participants may be found in [6].

In total, the study included 314 participants from 21 institutions (FC = 136, GS = 150, E = 28). The student participants were also assigned descriptors specifying their level of technical competence. The descriptors ranged from 1 being a Picasso to 5 being a failing student. The descriptors were accompanied by a protocol to determine the allocation of students to each category. Transcripts for each participant contributed to an average GPA calculation in Computer Science courses. On a four point (4.0) GPA scale (used in the USA), the divisions occur at: 4, 3.7, 2.7, 1.7 and 0 where these numbers represent the bottom of the category in question. A Picasso has a 4.0; a 3.7 or higher falls into the Top category. In terms of percentages, these would mean: 100%, 93%, 67%, 42% and 0. In terms of letter grades, this roughly maps to: A+, A, B-, C- and F. Here, a “D” was considered to be synonymous with failure (although this is considered a pass in some contexts). Note that the performance bucket classifications are in order from 1 to 5, with 1 being the most technically competent and a 5 being the least technically competent. All performance buckets contain at least one GS participant and one FC participant. Table 1 shows the number of participants in each performance category.

Table 1. Performance Bucket Percentages for FC and GS Populations

<table>
<thead>
<tr>
<th></th>
<th>1 (Picasso)</th>
<th>2 (Top)</th>
<th>3 (High)</th>
<th>4 (Low)</th>
<th>5 (Fail)</th>
<th>No Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC (N=136)</td>
<td>11%</td>
<td>18%</td>
<td>41%</td>
<td>19%</td>
<td>7%</td>
<td>4%</td>
</tr>
<tr>
<td>GS (N=150)</td>
<td>6%</td>
<td>23%</td>
<td>58%</td>
<td>13%</td>
<td>1%</td>
<td>0%</td>
</tr>
</tbody>
</table>

2.3 Data Analysis

To study participants’ recognition of ambiguity and the level to which they addressed requirements in the design brief, the following questions were asked of the data corpus:

1. Did the participant ask at least one question about ambiguities/omissions in the specification (as distinct from procedural questions and questions about word meanings)? (Yes or No)
2. Did the participant make at least one explicit assumption in the oral description, written representation, or other recorded responses about ambiguities/omissions in the specification? (Yes or No)
3. Did the subject address the requirements of the specification? (Yes, Partially, Hardly, No)

Nine distinct requirements were used when determining the answer to question 3. See Appendix A for a list of the requirements. If all nine requirements were addressed, the participant was deemed as “yes” for satisfying the requirements. If five to eight were satisfied, the participant was classified as “partially”. If one to four were satisfied, the participant was classified as “hardly” and if no requirements were addressed, the participant was classified as “no”.

3 Recognition of Ambiguity

We define two subpopulations for the purpose of classifying participants in the study. The classification is based on observable participant behavior that indicates whether or not the participant recognized ambiguity in the design brief specifications. The observable events we considered for evidence as recognizing ambiguity are question-asking and oral/written assumptions. A participant who asked at least one question about the specifications had recognized that the specification was underconstrained, had omissions, or needed further clarification. A participant who wrote assumptions or explicitly stated assumptions during the interview recognized that the specification was underconstrained and made assumptions to progress to the design phase.

The recognizers are participants who either asked a non-procedural question or made an explicit assumption in the design process. Both spoken and written assumptions are classified as explicit assumptions. The information gatherers, a subset of the recognizers, asked questions and may or may not made observable assumptions. A participant who made an assumption but did not ask a question was classified as a recognizer, but not an information gatherer. Table 2 shows the classification of recognizers and information gatherers based on the observable events of making assumptions and asking questions.

Participants who gathered information asked questions such as:
- Who decides if a student should get more sleep?
- What does it mean by 3 red alerts in a row? Over a period of days? 3 nights in a row?
- What happens when the student does not wake up with the alarm?
- Does the mother own a Pocket PC?
- Is the two minutes before or after the alarm is going off?
- How will the report to someone who cares be transmitted? Should it be email, a letter, or what?
Table 2. Criteria Used to Determine Recognizers and Information Gatherers

<table>
<thead>
<tr>
<th>Made Assumption</th>
<th>Asked Question</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Recognizer</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Recognizer</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Recognizer</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Non-Recognizer</td>
</tr>
</tbody>
</table>

- Is the device an aid or an enforcer? Do we have to worry about people lying? Should it be belligerent, annoying when people need sleep?

The participants' procedural questions that did not indicate recognition of ambiguity include the following:

- So you are really not looking for actual code, but just how you would go about doing this?
- So you are looking for a description in English, not in code?
- What are you looking for? ... do I need to write ... just ... an outline?
- So when you say design what do you want? I mean how much detail? You want a list of features? You want a basic design an overall system ... maybe a block diagram?
- How much detail do you want? Do you want pseudo-code?
- Can I use a calculator?

Spoken or written assumptions about the alarm clock system indicated recognition of ambiguity. Participants' spoken assumptions during the design process include:

- Average number of hours of sleep for each student would be 8.
- Made assumption about user always entering when they wake up and when they go to sleep.

**Issues**

- Will the red alarm go off if I have already set a "Sleep" alarm?
- Should this be counted sep?

Fig. 2. Example of recognition of ambiguity in design document

![Red alerts are reset if there is a day without red alert.](image)

- Red alerts are reset if there is a day without red alert.
- What I like is 7-8 hours of sleep - that is me personally. So I'm going to use that.
- Yellow alert to beep user after fourteen hours of wake up time, red alert to beep user after sixteen hours of wake up time.
- I assume that the PDA has yellow lights.
- I'm assuming Java since it is the only programming language I know.

Fig. 3. Example of recognition of ambiguity in design document in the form of written questions. See bottom of figure.
- It would be the "sleep time reminder". And, uh, that could be set for any
time during the day, as long as it was set for at least seven hours before the
wake time.
- umm, i was going to ask about the capabilities of a pocketPC, but I thought I
could just assume that is had sort of the capabilities of a handheld computer.

Participants could also express their recognition of ambiguity in the docu-
ments they created. Figure 2 shows an example of one participant who explicitly
wrote down the issues encountered while producing a design solution. Another
participant wrote down questions, shown in the lower half of Figure 2.

We recognize a source of bias in that our definitions of subpopulations are
based on observable behavior (asking questions and making explicit assump-
tions). Participants making assumptions without externalizing them are not de-
defined as recognizers in this work and, consequently, the number of participants
making assumptions may be larger than reported.

A second source of bias is introduced when determining if a participant's
verbal or written declaration represents a non-procedural question or an as-
sumption. In the few cases where the participant could not be classified, the
participant is coded as "Don't Know". One reason why a participant could not
be classified is due to illegible handwriting on design sheets, and a confident deci-
dion could not be made regarding the nature of the scribble as an assumption or
not. Due to the nature of the study, participants' interviews were tape recorded.
In some cases, the tapes are inaudible so determining whether a participant did
or not recognize ambiguity in spoken form was impossible. Each investigator
classified the participants that he or she interviewed. In most cases, the investi-
gator was an educator at the institution where participants were recruited. Consen-
census about what constituted a non-procedural question and written/spoken
assumption among all 21 investigators took place prior to classification.

4 Performance Analysis

Our data analysis includes a breakdown of recognizers and information gath-
ers by participant population (FC, GS, and E), the number of requirements
addressed in the design solution, and performance bucket. We also analyze the
time taken to perform the task. Of the 314 participants, 11 could not be classified
as recognizers or non-recognizers because of a "Don't Know" response from the
researcher reporting the data, so 303 participants were used in the data analysis.

By Population. Table 3 shows that a greater percentage of graduating students
recognized ambiguity than first competency students. The same can be said of
information gatherers. Educators overwhelmingly recognize ambiguity. Note
that 36% (78/216) of recognizers simply made assumptions without gathering
information. Of FC recognizers, 48% simply made assumptions. This is consistent
with the practices of beginning engineering students [5].

Results on the impact of experience (defined by population level) was statis-
tically significant for both recognizers and information gatherers. A chi-squared
test with a desired significance of 0.05 leads us to reject the following hypotheses:

- \( H_1 \): The distribution of recognizers is independent of experience (population)
level. \( p \leq 0.025 \)

and

- \( H_2 \): The distribution of information gatherers is independent of experience
(population) level. \( p \leq 0.01 \)

<table>
<thead>
<tr>
<th>Table 3. Recognizers and Information Gatherers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Percent Recognizers</td>
</tr>
<tr>
<td>Percent Information Gatherers</td>
</tr>
</tbody>
</table>

Addressing Requirements. Recognizers were also more successful at addressing
the requirements outlined in the design task. Each participant's design was cat-
alyzed by how well it addressed requirements: Yes (all 9 requirements ad-
ressed), Partially (5-8 addressed), Hardly (1-4 addressed), No (0 addressed),
NA (no data about requirements). Table 4 shows the performance of each group
in satisfying the requirements. Almost half of the recognizers addressed all the
requirements, while only 26% of non-recognizers did. The results for information
gatherers are similar.

Additionally, of those who addressed all requirements \( (N = 120) \), 81% were
recognizers and 52% were information gatherers. Of those who partially ad-
ressed the requirements \( (N = 151) \), 67% were recognizers and 44% were in-
formation gatherers. A weighted average was used to estimate the number of
requirements addressed by recognizers and non-recognizers. We assigned Yes
the value 9, Partially the value 6.5, Hardly the value 2.5, and No the value
0. Using these values as an estimate of the number of requirements addressed
by participants assigned to each category we found that recognizers addressed
an average of 7.3 requirements and non-recognizers addressed an average of 6.5
requirements.

The differences in rate of addressing requirements is statistically significant
when comparing recognizers versus non-recognizers and also when comparing
information gatherers and non-gatherers. A chi-squared test with a desired sig-
nificance of 0.05 leads us to reject the following two hypotheses:

- \( H_3 \): Recognizers and non-recognizers address requirements at the same rate.
(p \leq 0.01)

and

- \( H_4 \): Information gatherers and non-gatherers address requirements at the same rate. \( p \leq 0.025 \)

However, when comparing information gatherers versus the subset of recogni-
izers who do not gather information (i.e. those that merely make assumptions),
we cannot reject the following hypothesis:
Do Students Recognize Ambiguity in Software Specifications?

Table 4. Addressing Requirements By Population

<table>
<thead>
<tr>
<th></th>
<th>Yes</th>
<th>Partially</th>
<th>Hardly</th>
<th>No</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizers (N=216)</td>
<td>45%</td>
<td>47%</td>
<td>6%</td>
<td>17%</td>
<td>17%</td>
</tr>
<tr>
<td>Info Gatherers (N=138)</td>
<td>46%</td>
<td>49%</td>
<td>4%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Non_INFO Gathers (N=165)</td>
<td>35%</td>
<td>51%</td>
<td>13%</td>
<td>2%</td>
<td>6%</td>
</tr>
</tbody>
</table>

H5: Information gatherers and those who recognize but do not gather information address requirements at the same rate.

Performance Buckets. When looking at recognizers and information gatherers with respect to performance bucket (1 = Picasso ... 5 = Fail), one might expect that participants in the higher buckets would tend to be the recognizers and the information gatherers. However, our data do not support this hypothesis. Figures 4 and 5 show the distributions of recognizers and information gatherers across the five performance buckets. A performance bucket could not be determined for six FC participants.

Figures 4 and 5 provide a visual representation of the percentage of recognizers and information gatherers, respectively, in each of the five performance buckets. From these graphs, it is evident that there is little difference between the performance buckets, but there is a consistent difference between first competency participants and graduating seniors. Across all buckets, as shown in Table 3, the percentage of recognizers increases from 63% for first competency participants to 76% for graduating seniors. Similarly, the percentage of information gatherers increases from 33% for first competency participants to 50% for graduating seniors. The largest deviations in percentages of recognizers and information gatherers appear in performance buckets 1 and 5, but these deviations in percentages are probably artifacts of the sizes of these buckets rather than general trends.

What is more interesting is that the difference between first competency participants and graduating seniors is greatest in performance bucket 1 (Picasso). This could be a result of the size of the bucket. The size of the Picasso bucket for FCs is 15, and the size of the Picasso bucket for GSs is 9. Because FC participants have taken few Computer Science courses, the GPA measurement of FC participants may not be a representative measurement of performance. It could also be that beginning participants with exceptional skills are over confident. As these students mature and gain experience they might recognize the importance of addressing ambiguities in a design specification. Of the 15 first competency participants in bucket 1 only 4 asked questions about ambiguities and/or omissions in the specification. Among graduating seniors classified as Picassos, 5 out of 9 asked a question clarifying the specification. If one looks at participants in bucket 1 that asked a non-procedural question and made an assumption we find that 5 out of 9 graduating seniors fall into this category as compared to only 2 out of the 15 first competency participants.

Statistical tests on the distribution by performance bucket for either recognizers or information gatherers could not be found to be statistically significant. Specifically, chi-squared tests with a desired significance of 0.05 would not allow us to reject the following hypotheses:

H6: The distribution of recognizers is independent of performance bucket.

H7: The distribution of information gatherers is independent of performance bucket.

Time. The average amount of time taken to perform the design decomposition task (before talking about the design aloud) was 41.2 minutes for recognizers and 31.7 minutes for non-recognizers. Information gatherers took, on average, 44.0 minutes and non information gatherers spent 33.8 minutes on the design task. This result is not too surprising, since the act of recognizing ambiguity and asking questions takes time during the initial design phase. Figures 6 and 7 show the distributions of time taken by recognizers and information gatherers, respectively.

Per Institution. If we look at the percentage of recognizers and information gatherers, separated by institution we see that some institutions have a high number
Fig. 5. The percentage of information gatherers per performance bucket (1 = Picasso, 2 = Top, 3 = High, 4 = Low, 5 = Fail). The absolute numbers of participants falling into each performance bucket are listed above the bars.

Fig. 6. Distribution of time taken to complete design decomposition task, separated into recognizers and non-recognizers.

Fig. 7. Distribution of time taken to complete design decomposition task, separated into information gatherers and non-information gatherers.

Fig. 8. Breakdown of information gatherers and recognizers by institution. Each institution is labeled by a letter and a number on the x-axis. The number indicates the total number of participants interviewed at the institution.

of recognizers. Five of the 21 institutions had all participants who recognized ambiguity. The data in Figure 8 indicate that all institutions had at least one recognizer and at least one information gatherer.
5 Discussion and Future Work

This work describes a multi-national, multi-institutional study of student designs which reveals that recognition of ambiguity is more common among graduating seniors than first competency students and that recognizers tend to address more requirements in the design. The majority of participants (81%) who addressed all requirements in the design also recognized ambiguity. Additionally, those participants who recognized ambiguity addressed an average of 7.3 requirements while non-recognizers addressed an average of 6.5 requirements. The result of more seniors recognizing ambiguity is consistent with the findings of Atman et al. [5]. In our study, 48% of the first competency students made assumptions without requesting additional information, while 34% of graduating seniors fell into this category. (These participants were recognizers but not information gatherers.) Making assumptions without gathering information appears to be a characteristic of less experienced designers.

The study presented here categorized participants as recognizers and information gatherers. Table 2 indicates the criteria for determining recognizers and information gatherers. We did not attempt to distinguish information gatherers who did not make assumptions versus information gatherers who did make assumptions for most of the analysis presented in this paper. Future analysis using all four categories in Table 2 may reveal additional trends among the FC, GS, and E subpopulations.

The study included three metrics for comparing participants and their designs. The first metric, performance buckets, affords a comparison of participants from different institutions and is independent of the software design produced by the participants. The other two metrics, number of requirements addressed and time needed to complete the task, are directly related to the design task but do not convey the quality of the design. At this point, no attempt has been made to assess the quality of participants’ designs, but others have attempted to compare designs based on formal notation, grouping and interaction of parts, communication between parts, and hierarchical organization. More details can be found in [6] and [14]. These other metrics could prove useful for studying recognizers and information gatherers.

5.1 Research Study Design

The study investigation took place at 21 different institutions, requiring explicit instructions for recruiting participants, receiving human subjects approval, gathering data, and performing data analysis. The investigation was a collective research component of the Scaffolding Research in Computer Science Education, a NSF-supported project to train Computer Science faculty in educational research methods [1]. The overall investigation posed the questions: Can computer science students decompose a problem into parts? and What criteria do students use when evaluating and producing designs? Because the questions are broad, the authors narrowed the focus to students’ recognition of ambiguity in the alarm clock description. In the data collection instructions, the researchers were asked to instruct students back to the alarm clock design brief if they had any questions about the specification and to make any necessary assumptions. Had the study investigation focused on the recognition of ambiguity from its inception, data specific to our research questions could have been collected.

The study investigation took place at 21 different institutions, requiring explicit instructions for recruiting participants, receiving human subjects approval, gathering data, and performing data analysis. The investigation was a collective research component of the Scaffolding Research in Computer Science Education, a NSF-supported project to train Computer Science faculty in educational research methods [1]. The overall investigation posed the questions: Can computer science students decompose a problem into parts? and What criteria do students use when evaluating and producing designs? Because the questions are broad, the authors narrowed the focus to students’ recognition of ambiguity in

5.2 Implications for Educators

This work raises some important questions for educators. Recognition of ambiguity and information gathering appear to be independent of a student’s academic achievement. It is our hope that these questions will be discussed and fleshed out by interested investigators and will lead to new studies and interventions to better understand the possibilities and benefits of addressing ambiguity in the undergraduate curriculum.

The recognition of ambiguity in underconstrained design environments is an important skill in the design process. Our research study shows that a higher percentage of graduating seniors recognize ambiguity than do first competency students. This could be a result due to the educational experience of the participants. Perhaps first competency students are provided with fully specified problems to solve, while upper-division students are more likely to solve underconstrained problems. In many Computer Science curricula, upper-division students have the opportunity to take capstone courses, independent studies, and industry-partnered courses where underconstrained design problems are likely to occur. A recommendation based on the results of this study is that educators provide and discuss strategies for handling underconstrained design problems at all levels in the curriculum. If the initial design phase of problem-solving is not explicitly addressed in introductory computing courses, educators may want to include such a discussion in the course and provide activities for students to practice recognizing ambiguity.

Participants who recognize ambiguity tend to address more design requirements. We did not attempt to classify participants’ designs with regard to quality, but making an attempt to address most or all requirements is a quantifiable measurement necessary for high-quality designs. A recommendation to educators is to include a specific mapping phase in the design process where students can clearly link each design element to a specific requirement in the specification.
The recognition of ambiguity does not appear to reflect past performance, as measured by grades. One interpretation is that grades themselves may not measure students' skills in recognizing ambiguity, as this part of the design process may not be included in the overall grade assigned to homework assignments. Additionally, first competency students usually had recorded grades for just one to three computing courses; therefore, FC grades may not accurately indicate design skill. A recommendation to educators is to formally assess design process elements, such as recognizing where more information is needed to carry out a design, in addition to the final product.

Stepping back from our role as researchers, we observed the utility of the design brief exercise and the design criteria task as useful activities in the classroom. It was quite illuminating for us to see and hear students describing their designs and their design process. Because we were in the “researcher” role and not the “educator” role, we could focus on students’ processes rather than the products. As educators, much of what we can “see” of students’ process is during office hours, when they usually have questions, and in documentation of the final product. Some students tried to write code for their designs while others simply described which component would satisfy each bullet point in the design brief. Once students completed the decomposition task for the alarm clock, they described “parts” of their design to us. Their definitions of what a part is in software design may be of interest to their instructors. Educators may want to use such a design exercise to learn about how students interpret a specification and produce an initial design.

5.3 Future Work

The results of the study raise new questions for future investigation, both informally as action research for Computer Science educators and more formally as multi-national research studies similar to the one described in this paper.

- What can we do to help students recognize ambiguities in specifications?
- How should students be taught to resolve ambiguity in specifications? How should we teach them to gather information and how does that impact our traditional evaluation mechanisms?
- At what point in an undergraduate curriculum should we intentionally introduce ambiguity into problem statements? Brown points out the benefits of having students in CSI write formal program specifications [4]. Is teaching students how to recognize and respond to ambiguity in the first year feasible or advisable? Is it so important that we address it in the first class, or will that only add to confusion?
- Will addressing the issue of ambiguity more directly in instruction really improve students’ design abilities? Put another way, is becoming a better designer a matter of maturity, an intrinsic ability, or something that can be taught and learned?
- Are there exemplary curricula that help students learn to recognize and resolve ambiguous specifications?

Acknowledgments

We thank the participants and leaders of the Scaffolding Workshop for CS Education Research for their help in data collection and analysis. We especially thank all the students and educators who participated in the study. 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.

References

Appendix A

Requirements Used for Determining Satisfaction of Requirements

1. Allow a student to set an alarm to wake them up.
2. Allow a student to set an alarm to remind them to go to sleep.
3. Record when a student tells the system that they are about to go to sleep.
4. 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).
5. Make recommendations as to when a student needs to go to sleep. This should include yellow alerts when the student will need to sleep soon, and red alerts when they need to sleep now.
6. 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.
7. 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.
8. 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.
9. Wake the student up.