Senior Software Engineering Students' Understanding of Design

Dr. Andrea L. Schuman, California Polytechnic State University, San Luis Obispo

Andrea Schuman is an assistant professor in the Department of Computer Engineering at Cal Poly. She holds a Ph.D. in Engineering Education and an M.S. in Electrical Engineering from Virginia Tech, and a B.S. degree in Electrical Engineering from the University of Oklahoma. Her research interests include experiential teaching and learning in ECE and global engineering.

Dr. David B Knight, Virginia Polytechnic Institute and State University

David Knight is a Professor in the Department of Engineering Education at Virginia Tech and also serves as Chief of Strategy in the College of Engineering and Special Assistant to the Provost. His research tends to be at the macro-scale, focused on a systems-level perspective of how engineering education can become more effective, efficient, and inclusive, and considers the intersection between policy and organizational contexts. Knight currently serves as the co-Editor-in-Chief of the Journal of Engineering Education.

Mohammed Seyam, Virginia Polytechnic Institute and State University

Mohammed Seyam is a Collegiate Associate Professor in the Computer Science Department at Virginia Tech. He is a researcher and educator in the fields of Software Engineering, Human-Computer Interaction, and Computer Science Education. Additionally, he is the CS Department Coordinator for Experiential Learning, where he leads several initiatives to enhance students' learning through out-of-classroom experiences, including the CS Study Abroad program. Mohammed has 20+ years of experience in teaching university level courses, and he presented and conducted multiple talks and workshops in different countries. Among other courses, he taught: Software Engineering, Database Systems, Usability Engineering, and Software Project Management.

Senior Software Engineering Students' Understanding of Design

Abstract

Design thinking is an important skill for computer science students because the software engineering field requires professionals to solve open-ended problems. Software Engineering capstone courses aim to teach design thinking to prepare graduates with the skills to work in different environments. Capstone experiences, specifically, build upon prior theoretical learning so students can practice a software-agnostic process and create solutions with a human-centered focus. Prior research indicates that there is a gap between design in software engineering education and industry, so it is valuable to explore how students understand design thinking. The purpose of this qualitative study is to explore the conceptions that Computer Science students have about design at the beginning of their senior capstone. The participants in this study are 31 students majoring in Computer Science, Secure Computing, or Data-Centric Computing and enrolled in a Software Engineering Capstone course. Students recorded themselves speaking based on a series of reflection prompts. In their first reflection, which is the focus of this paper, the students were asked "How would you define design in computer science?" The transcripts of their responses were analyzed using provisional coding based on prior definitions of Design Thinking from literature. Their responses were used to answer the research question: *How do software engineering undergraduate students define design at the beginning of a capstone course?* The results of this study indicated that about half of the students demonstrated an understanding that design involves both planning and implementation, though some stages were underrepresented in their responses. This analysis illuminates gaps in knowledge from prior experiences that capstone instructors should focus on covering.

Introduction

Software engineering degree programs need to prepare students with both theoretical foundations for the field and practical experiences so that they can apply their computer science skills [1]. The IEEE Computer Society [2] emphasizes design skills as a priority for software engineers to be able to create software that can solve problems. Agile processes are one common project management framework. It is a methodology specifically used to create solutions in software engineering. Agile involves collaboration with customers to understand their needs, responding to changes, re-releasing software with updates, and iterating until the final product [1]. Design thinking is a more common problem-solving approach in the broader discipline of engineering. Design thinking can be viewed as a toolbox, a process to follow, or an overall mindset [3]. As a process, design focuses on the context and users' needs, finding creative solutions, and iterating prototypes [3]. Both agile and design thinking are collaborative methods to create solutions to a problem. Agile tends to involve quick iterations and incremental improvements and design thinking has more emphasis on spending time thoroughly understanding the user's needs [4].

Regardless of the method of implementation, sufficient time and planning needs to be dedicated to the design stage or chaotic, inefficient code can be created. Technical debt, defined as fundamental issues in the code base, can accrue and it must be corrected to create a robust system [2]. Software engineering capstones are project-based experiences that prepare students with design skills that are agnostic to the technology that is used. The purpose of this qualitative

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 2235205. Any opinion, 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.

study is to explore the conceptions that senior-level Computer Science students have about design when they begin their capstone experience through the lens of engineering design thinking. By better understanding what students' prior knowledge is about design, capstone courses can build on prior mental models and fill gaps in knowledge.

Research Question

How do software engineering undergraduate students define design at the beginning of a capstone course?

Literature Review

Chong et al. [5] examined a similar research question to the current study, except with first-year engineering students as their population. This study found that first-year students' initial journaled definitions of engineering design were relatively similar to textbook definitions, with a limited vocabulary and a focus on solving a problem. After a course design activity, the students gave another definition of design. The follow-up definitions included more diverse objectives of engineering design such as safety and environmental causes. For both definitions, a significant portion of students did not see engineering design as a process to follow, but instead as a descriptor of the result of the process. This study demonstrated the difficulty of understanding engineering design before someone has experienced it [5]. The current study has senior-level students as its population, who we can expect have experience with design from prior courses and internships in industry.

Dobrigkeit and de Paula [3] implemented case study research at a global IT company that promotes design thinking. They observed and interviewed three teams of software developing professionals to learn how they used design thinking and agile. The research indicated that the three main perspectives on design thinking were to view it as a mindset, process, or toolbox (or combination). The results varied between professional roles. Those with more experience in design tended to view it as a mindset. The study overall found that design thinking could be used in software engineering to explore users' needs, which can align with the requirements-setting process in agile methods. If a design-thinking mindset is developed, the problem-solving methods can be applied beyond engineering projects. Overall, their findings supported previous work that characterized design thinking as the overall steps to find a creative and user-centered solution, with agile methods and the step-by-step process to implement the solution [3].

Palacin-Silva et al. [6] created a software engineering capstone course to specifically combine design thinking with agile methods. Their motivation was to address the gap between how students are taught design thinking and how it is applied in the software industry. They defined design as a process of cyclically going through three stages: inspiration, ideation, and implementation. The authors implemented these steps into the course with the software design techniques of personas, storyboards, journey maps, prototypes and usability testing. Their assessment showed that students learned about the importance of prototyping and iterating based on feedback [6].

Corral and Fronza [1] compared two software engineering undergraduate courses that were aligned with either agile methods or design thinking. As a part of their research, they created a framework to map agile practices with design thinking steps, which is shown in Table 1.

Table 1 Mapping between Agile Practices and Design Thinking, from [1]

Agile Practices	**Design Thinking**
Analysis: Lean management, system metaphor, user stories	*Empathize:* Market research, interviews, user persona, user journey
Design: Agile modeling, CRC Cards, story-driven development, customer on site	*Define:* Point-of-view statement, Analysis/synthesis, problem statement
-	*Ideate:* Brainstorming, point-of-view analysis, How might we templates
Implementation: Backlog management, Extreme Programming, Feature-driven development	*Prototype:* Storyboard, sketching, non-functional prototyping, functional prototyping
Testing: Test-driven development, continuous integration, sprint retrospective	*Testing:* Minimum viable product rollout, user feedback grid

The researchers compared the course artifacts and projects and found that the agile course had more software-centric assignments that had technical, high-quality software. The design thinking course solutions were more human-centric and focused on solving the given problem [1]. This research's mapping is used as the theoretical framework for the study.

Theoretical Framework

In this study, the theoretical framework of Corral and Fronza's Mapping Between Agile Practices and Design Thinking [1], shown in Table 1, is the basis for the codebook used to analyze the data. Students' responses were coded based on which practices or steps were included in their definition of design in software engineering. The definitions in the codebook were synthesized from the rest of Corral and Fronza's paper, with additions from the authors [1]. The codebook for the current study is shown in Table 3.

Research Methods

The research methods for this study are qualitatively analyzing capstone students' reflections in which they are answering *"How would you define design in computer science?"* The responses are transcribed and analyzed using the theoretical framework of Corral and Fronza's Mapping Between Agile Practices and Design Thinking [1]. The results are frequency counted. Following are more details on the data collection, study participants, and data analysis that were used with the goal of answering the research question: How do software engineering undergraduate students define design at the beginning of a capstone course?

Data Collection

Throughout the capstone course, students recorded themselves speaking based on reflection prompts. Spoken reflections were chosen to gain an understanding of students' in-the-moment thoughts without editing. They can be completed quickly and can be easier for students who

struggle with writing. Students were required to complete the reflections to receive points towards their grades, but they chose whether or not to consent to have their data used for research. The instructor was not informed of their choice to consent for research, and this choice did not impact their grades. Eleven students declined to participate in the study and seven students who consented to participate did not submit the first reflection, which was the data selected for the current study. The prompt was:

In the first reflection, the research team is hoping to gain an understanding of your background coming into this course.

1. *How would you define design in computer science?*
2. *What are your previous experiences with design work?*
3. *What are your goals from this course?*
    - *3.1. What do you hope to learn from this course?*
    - *3.2. How does this course align with your plans for the future?*

The sections of the transcripts of the participants' responses in which they answer question 1 were for this study's analysis. The answers to question 2 were additionally categorized to identify the frequency that students discuss coursework, internships, and extracurricular activities as design experiences in the Participants section. The responses to the first reflection were typically two to four minutes long.

Participants

The participants in this study are 31 undergraduate students who are enrolled in a Software Engineering Capstone course. Their majors are Computer Science, Secure Computing, or Data-Centric Computing. Computer Science is the most common major. In total, 49 students were enrolled in the capstone course, which took place at a large, research-intensive public institution.

To better understand the current population of students, their answers to the question *"What are your previous experiences with design work?"* in Reflection 1 were examined by the authors to see if students discussed Coursework, Internships, or Extracurricular Experiences. If students discussed multiple types of experience, the student was added to each category's count. It is worth noting that this may not capture all of the students who previously had internships or extracurricular experiences, if they did not view these as "design work". All respondents gave an answer that included at least one of the three categories of design experiences. The full results are in Table 2, which shows the number of students that discussed each type of experience in their answer.

Table 2 Overview of Students' Design Experiences

Prior Design Experiences	**Coursework**	**Internships**	**Extracurricular Experiences**	**Total Participants**
Number of Students	26	17	6	31

Coursework was discussed by the majority of students: 26 of 31. Courses that were frequently referenced as teaching design skills included the following disciplinary courses: *Software Design*

& Data Structures, Introduction to Human-Computer Interaction, and *Data Structures & Algorithms.* These courses had hands-on group projects and were associated with design skills by the students. Other courses in the curriculum that were referenced less frequently included disciplinary electives and the *Engineering Foundations* courses, which are required for all engineering majors and include an overview of engineering design and problem-solving.

Seventeen of 31 (55%) students said that their previous experiences with design work included an internship. The data indicated that students were drawing on these experiences in their definitions, such as one participant's response of "before my previous experiences, I thought of design as something that you would just do in class, and not really have to deal with right after. That's like most things in school. But I was kind of surprised to see it being used in a professional setting. It showed me that [design is] important later on… I've seen how companies and industries use design to help them improve their system, to help them get to know what the client wants and what their needs [are]." This quote demonstrates that students are drawing on their experiences beyond the classroom when defining design.

Extracurricular experiences were referenced by six students. These included university design competition teams, hackathons, and personal programming projects.

Data Analysis

To characterize the students' definitions of design, we used structural coding. Structural coding focuses on the content of the data through the lens of an a priori codebook [7]. In this study, each response was coded to see if it included the following design practices: Empathize, Define, Ideate, Prototype, and Test, which are defined in Table 3 [1]. After the responses were coded, each code was frequency counted and the combinations of codes in responses were examined.

Table 3 Codebook for Current Study

Codes	**Definition**
Empathize	Understanding the users, the problem, and the feasibility of solutions.
Define	Defining the needs of the customer in terms of a problem that can be solved. Planning and making decisions to go from high-level ideas to practical implementation.
Ideate	Brainstorming and finding possible approaches to solve the problem, with ideas for implementation. The proposal of a solution (product, service, experience) that meets the needs of the customer.
Prototype	Implementing ideas to create a working product that covers a selection of the features required by the user.
Test	Checking if the design works. In software it is often concurrent to implementation

Research Quality and Limitations

The participants in this study are enrolled in multiple sections of a capstone class at a single university. The results are not broadly generalizable but do have transferability because this institutional context (i.e., large, research-intensive university) is similar to other institutions that educate large numbers of computer science students and because of the prevalence of industry

internship experiences that informed students' perceptions of design. Prior research has shown that nearly 60% of fourth-year computer science students have held an internship, and during that time students are learning about software design beyond their specific university's curriculum [8]. This population's answers were aligned with the prior research, since 55% of students said that they previously had design experience from an internship.

For research quality, one author created the initial codebook. They then asked a Computer Engineering industry professional to review the codebook. The author who created the codebook coded the responses and provided examples for codebook review in a multiple peer audit, with the previously mentioned Computer Engineering industry professional and two faculty members in Computer Science and Engineering Education faculty (who are the co-authors). Data that were difficult to characterize were thoroughly discussed until an agreement was reached so that the codebook's application would be consistent [9].

Results

This section includes an overview of the ways in which software engineering undergraduate capstone students define design at the beginning of their capstone course. The summary of results is included in Table 4. This table includes the number of participants that had each code appear in their response. If their response had more then one code, the student is counted in each code's category.

Table 4 Results by Code

Code	**Empathize**	**Define**	**Ideate**	**Prototype**	**Test**	**Total Participants**
Number of Students	9	16	12	23	7	31

Empathize

Empathizing is spending time to understand the problem space or customer in depth. The equivalent agile practice is Analysis [1]. Nine students included Empathize in their definitions of design. An example quote is "[Design is] basically everything that occurs that needs to be done before the actual implementation of whatever you're trying to build. And that's understanding the requirements for this product, talking to the end users and understanding what they need to get out of the product you're trying to build, and what's a problem they face that you can solve." In this student's response, they frame the problem space around requirements, which is aligned with computer science design methods. Similarly, another student discussed both users' needs and project requirements, "I would define design in computer science as being able to be given... project specifications or client needs, and be able to really take in what problem they are having in the moment." Based on requirements, the participant said that it is their job as the engineer to understand the details of the problem.

Define

Defining the problem involves setting the benchmarks and outcomes for the project. The understanding of the users' needs is translated into goals that will be accomplished by the technology, and it corresponds with Design in agile. Define was the second-most common code, and 16 participants included it. One quote is "I believe design in computer science is creating…

a set of criteria and requirements." This definition precisely aligns with Define steps. Another said in more detail, "describing what you want your system to do... [and] some kind of idea of what you expect it to look like or how you expect it to work." By setting the expectations for the outcome of the software engineering project, they would be in the Define stage.

Ideate

Ideation is the step where engineers brainstorm and propose possible solutions based on their understanding of the problem space. Corral and Fronza did not have an equivalent step for Agile methods [1]. Twelve of the participants included Ideate as a step, and one student said, "The process of coming up with an approach to a particular issue or solving a particular problem...there could be multiple designs to a problem based on the constraints and needs." The existence of a variety of solutions for the problem is a hallmark of design thinking. Another student described "design is the process that comes when you're brainstorming potential solutions". In previous courses, some assignments required a minimum number of brainstormed ideas before students began implementing their projects.

Prototype

Prototyping is the beginning of the implementation phase. It was the most commonly appearing code in the data, and 23 participants' answers were coded with Prototype. The equivalent of Prototype is Implementation in agile, which is also a step of engineering design thinking [1], [6]. One example of Prototype in a response is, "pseudocode or how you define each method will go about what each method will do". Pseudocode is a simplified version of what will be written in code, and a way to prototype software projects. Another student said "You need to have your structure set. The algorithms in which you're building things, those are all there." Any stage of early programming is equivalent to prototyping in engineering, and this quote shows the usage of algorithms in early programming.

Test

Testing is examining and updating the software based on engineering issues or user feedback. It is the same step in agile methods. Test was the code with the fewest appearances, with seven instances. An example code is "most of [design] is iteration, or testing as well where you can learn off of your previous mistakes." Testing is frequently done concurrently to programming, but students did not often explicitly include it in their answers. When it was discussed, optimization and efficiency were frequently cited as the goals for improving the project. One student said, "being able to identify and optimize various aspects of your application to make sure everything is as smooth as possible." To optimize a project, testing and iteration is necessary.

Summary

The responses ranged in their thoroughness. One student's response had none of the five codes, and another included all five. The response with all five steps is,

> "I would usually try to understand the requirements or the proposal first. This would be reading through the specifications and just gathering my knowledge on what I already know... I would collaborate with others on their ideas and how they would try to understand the project specifications or how they would approach their solutions and then that's where we would finalize a possible solution. And so once enough understanding of the project is there, I start by creating the foundation, or the structure...I usually test as I go and then build off of the current structure until I'm satisfied with the end product."

This example showed the entire design process from understanding the problem space through testing and iteration.

Discussion

To gain an understanding of how the codes were associated within students' responses, we segmented the codes between Planning and Implementation. The Planning codes are Empathize, Define, and Ideate, and the Implementation codes are Prototype and Test (Table 5). If an answer had a code from each code group, their response was categorized under Both Planning and Implementation.

Table 5 Planning and Implementation Responses

Code Theme	**Planning (Empathize, Define, Ideate)**	**Implementation (Prototype, Test)**	**Both Planning and Implementation**	**Neither Planning nor Implementation**
Number of Students	6	9	15	1

As shown in Table 5, 15 of the 31 responses included at least one code from both Planning and Implementation. This result demonstrates that a significant percentage of students understand that the definition of design is not limited to one stage of the process. The capstone students in this population are near graduation and have had previous design experiences, which Chong et al. [5] argued is vital to understanding the complexity of design. The frequency of the inclusion of both codes and the more detailed responses than Chong et al.'s [5] data corroborates the finding that students are finishing their programs with a better understanding of engineering design than when they began the program.

Nine students' responses were focused on implementation. This finding may be because of computer science students' familiarity with agile processes, which require less time spent on planning. Most of the work is done in iteration and feedback [4]. Some students in this group included a mention of planning in general terms but not related to understanding the problem space. An example response is, "I would say design and computer science is essentially like a blueprint of your project. You want to have a good overview understanding of what you want to make or build before you do it. Because if you try to go straight into coding and just like figure out as you go, the project is going to get really messy… You can definitely adjust as you go. But it's not the same as having zero idea or goal of what you want your original project to go towards." Even though this student discusses planning in the abstract, it is not related to Empathize, Define, or Ideate. The response is focused on the Prototype and Testing stage.

Six students focused on the Planning portion of design, which includes both Inspiration and Ideation in three-staged engineering design thinking [6]. Engineering design is focused on the problem space more than the requirements focus of Agile processes [4]. The software engineering students in this study primarily framed planning around requirements. An example response is, "[design is] everything that needs to be done before the actual implementation of whatever you're trying to build. And that's understanding the requirements for this product. Like

talking to the end users and understanding what they need to get out of the product you're trying to build." This response specifically states that design is the stage before implementation.

The codes in the current study were focused on the stages of the engineering design process. Dobrigkeit and de Paula's [3] research found that software professionals can characterize design as a mindset, process, or toolbox. Some students' responses characterized design as completely open-ended. One example quote is, "I think most, if not all, graffiti artists have their own distinct unique style that can be easily distinguished amongst other artists. So I think this relates to computer science in that most computer scientists have their own distinct way of going about all of their tasks." This response is more aligned with the toolbox approach, in which portions of design thinking can be used ad hoc and uniquely, instead of systematically [3].

The word "design" in computer science is a semantically overloaded term. Other usages include front-end/back-end design, user interface (UI) design, and graphic design. These additional aspects were frequently mentioned by students, though most of them went on to additionally address engineering design thinking. An example response that addressed the multiple usages is "if you're in a front end design field, design would be more focused on maybe the user experience in the UI. But if you are more of a back end programmer, you would be more focused on the actual structure of the code and comments and how well designed the code is." The responses about laying out the structure of the project fit the design process coding better than the UI responses.

Conclusions and Implications

The results of this study indicate that a significant portion of senior software engineering students understand that design involves both planning and implementation. As compared to the results from Chong et al.'s [5] data, the senior students in the current study included more actionable steps in their definitions of design, as opposed to adjectives describing the process, which is more transferable to workplace projects. The steps of the engineering design process are not part of the curriculum, but most students understand at least some of the stages. One student included all five codes in their definition of design. To prepare students to enter the workforce, instructors should focus on filling the gaps indicate in the results. That included emphasizing the least common codes, which were Empathize and Test. Empathizing and understanding before moving to active planning is not a common stage in courses where the instructor sets the requirements but should be in a capstone course. Students are likely experienced in testing their code, but they can learn from the capstone course that it is a part of the engineering design process. The capstone course may also be the first time students are coordinating with an external stakeholder. In the capstone course, instructors could require more time spent understanding the stakeholder's requirements and feedback, which could correct Empathize and Test as the least common codes. In earlier courses that already have hands-on projects, instructors could include more in-depth assignments about understanding users' needs and setting requirements based on these needs, while making it clear that this Empathizing and Defining stage is part of design.

Since the current study investigates students' pre-capstone perceptions, future research could compare the answers to the same question pre- and post-course interventions. This paper's research design is a model that could be implemented by other instructors who want to examine

their students' understanding of engineering design. Students' learning and competency development during the current case will be published in more detail in the future.

References

[1] L. Corral and I. Fronza, "Design Thinking and Agile Practices for Software Engineering: An Opportunity for Innovation," in *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, in SIGITE '18. New York, NY, USA: Association for Computing Machinery, Sep. 2018, pp. 26–31. doi: 10.1145/3241815.3241864.

[2] "Why Software Design Is Important," IEEE Computer Society. Accessed: Jan. 02, 2025. [Online]. Available: https://www.computer.org/resources/importance-of-software-design-is-important/

[3] F. Dobrigkeit and D. de Paula, "Design thinking in practice: understanding manifestations of design thinking in software engineering," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, in ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, Aug. 2019, pp. 1059–1069. doi: 10.1145/3338906.3340451.

[4] K. Wangsa, R. Chugh, S. Karim, and R. Sandu, "A comparative study between design thinking, agile, and design sprint methodologies," *Int. J. Agile Syst. Manag.*, vol. 15, no. 2, pp. 225–242, Jan. 2022, doi: 10.1504/IJASM.2022.124916.

[5] A. Chong, J. A. Foster, P. K. Sheridan, and R. Irish, "Define 'Engineering Design': Understanding how freshman students develop their understanding of engineering, design, and engineering design," presented at the 2013 ASEE Annual Conference & Exposition, Jun. 2013, p. 23.365.1-23.365.24. Accessed: Sep. 18, 2024. [Online]. Available: https://peer.asee.org/define-engineering-design-understanding-how-freshman-students-develop-their-understanding-of-engineering-design-and-engineering-design

[6] M. Palacin-Silva, J. Khakurel, A. Happonen, T. Hynninen, and J. Porras, "Infusing Design Thinking into a Software Engineering Capstone Course," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, Nov. 2017, pp. 212–221. doi: 10.1109/CSEET.2017.41.

[7] J. Saldaña, *The Coding Manual for Qualitative Researchers*, Second edition. Los Angeles: SAGE Publications Ltd, 2012.

[8] A. Kapoor and C. Gardner-McCune, "Exploring the Participation of CS Undergraduate Students in Industry Internships," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, in SIGCSE '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 1103–1109. doi: 10.1145/3328778.3366844.

[9] C. Robson, *Real World Research: A Resource for Users of Social Research Methods in Applied Settings*. Wiley, 2011.