POGIL-like learning and student's impressions of software engineering topics: A qualitative study

Bhuvana Gopal
School of Computing
University of Nebraska-Lincoln
bhuvana.gopal@unl.edu

Ryan Bockmon
School of Computing
University of Nebraska-Lincoln
ryan.bockmon@huskers.unl.edu

Stephen Cooper
School of Computing
University of Nebraska-Lincoln
stephen.cooper@unl.edu

Abstract

In this study, we analyze students' impressions and perceptions of professional software engineering topics and practices. We explore student thoughts on the interconnections between various industry relevant software engineering topics such as requirements analysis, UI/UX, work patterns, agile communication, documentation, and business value. We studied student voices in a semester long undergraduate software engineering course, after they underwent instruction using POGIL-like, a guided inquiry based pedagogy. At the end of the course, we collected student responses to open ended questions regarding their perceptions of professional, end user software engineering topics. We combined student responses to these questions with researcher memos as well as reflective researcher journals. We analyzed these written responses using content analysis and identified the themes that the data yielded. The main themes that emerged from our qualitative analysis were: 1) Software is more than just a tool to solve business problems. 2) The need for documentation varies based on the process model adopted. 3) Timely and frequent communication between team members and stakeholders is essential. 4) Downtime during work is best used to further the sprint goals of the team or improve one's technical acumen. We attempt to understand if POGIL-like helped students develop a professionally sound understanding of basic software engineering topics and how they work to get a software product developed, from requirements to release.

1. Introduction

Student perceptions of a discipline are closely tied to their motivation to do well in it. When students feel comfortable with the environment they are in and the abilities they need to study the discipline, they tend to remain in the program more often than not. Retention is closely tied to motivation (Tinto, 1997; Bean, Eaton, et al., 2000). A good understanding of student perceptions of the social and academic experiences in software engineering can help improve student motivation in the discipline.

In an attempt to understand how CS students perceived various topics in software engineering (SE) and how they saw themselves in the discipline, we conducted a qualitative study with 24 sophomore/junior students with varying computing majors. We used POGIL-like (our implementation of Process Oriented Guided Inquiry based Learning) as the pedagogical approach. Our goal was to understand if students were able synthesize concepts from SE topics beyond just a basic level understanding of each topic, as well as how they perceived themselves in the discipline of software engineering.

The rest of the paper is organized as follows. We present prior work in software testing, POGIL-like, and student reflections in Section 2. We present our research question in Section 3. Our research methods are explained in Section 4. Themes from our data analysis are presented in Section 5, with a detailed discussion in Section 6. We detail the threats to the validity of our study in Section 7, and conclude in Section 8.

2. Prior Work

There are several studies on students' motivation and perceptions across various fields, and specifically in the STEM disciplines including CS. Some studies involve pedagogical interventions that have shown to affect student motivation as well.

Ames and Archer (Ames & Archer, 1988) showed that students' perceptions of classroom climate were related to specific motivational variables that have implications for the development of self-regulated learning as well as a long term involvement and interest in learning. Biggers et al. (Biggers, Brauer, & Yilmaz, 2008) discuss ways to keep student interest in CS, and strengthen their experiences. One of their key takeaways that an asocial, community-absent environment with limited human interaction among peers was a deterrent in students staying motivated and interested in CS. They also found that several students found CS work to be time consuming, boring and tedious, with a pronounced lack of social interaction. Students also had little to no idea of what a CS career looks like. This led to students dropping out within a relatively short period. Agosto (Agosto, Gasson, & Atwood, 2008) measured students' perceptions, attitudes, self-efficacy, and identity with respect to their intentions to further pursue computer science. In their study, self-perception regarding one's own ability to use CS techniques for problem solving, and identity (whether students thought they were computer scientists) emerged as the primary driver for differences in intention.

Peters (Peters & Pears, 2013) constructed a theory based framework to study students' CS identities, how students perceive learning CS and IT as meaningful, and how they are supported or hindered by their education. They found many students are influenced by their prior experiences with computing as far as their CS identity. Social interactions and group experience participation play an important role in shaping the CS identity of students. Dempsey et al. (Dempsey, Snodgrass, Kishi, & Titcomb, 2015) studied how to recruit and retain women in CS. They found that an increased CS self perception, specifically in terms of students' perceived self ability to be a computer scientist was a driving force in who stayed in CS.

Souza (Souza, Moreira, & Figueiredo, 2019) investigated students' perception on the use of problem-based learning (PBL) in an introductory SE course and found that there is a positive perception of the contribution of the project assignment on learning specific SE topics (such as software requirements, software Design, and agile methods), and this perception was even more positive for the students in the PBL course. Melnik (Melnik & Maurer, 2005) explored student perceptions of agile methods. Their experiences introducing agile methods in the CS courses indicate that students are enthusiastic about core agile practices and that there are no significant differences in the perceptions of students of various levels of educational programs. Almulla (Almulla, 2020) studied PBL in relation to students' motivation regarding their development of feelings of autonomy, competence, and relatedness (Almulla, 2020). They found a significant correlation between PBL and student engagement.

As seen from the studies above, there are research questions regarding student perceptions and motivation in CS and SE that have been answered. This is still a fledgling field, with many aspects yet unexplored. There are not many studies on how students perceive SE topics especially in the light of collaborative and active learning pedagogies, and through our study, we hope to fill the gap.

3. Research Question

In this paper we study students' perceptions of agile SE topics, within the context of a POGIL-like software engineering course. Our research question for this study was:

RQ: Were undergraduate students able to learn from and build upon multiple relevant concepts to display a connected understanding of software engineering topics when instructed using a POGIL-like pedagogy?

4. Method

4.1. POGIL-like: The approach

We chose to implement POGIL in the software engineering classroom since it is a process heavy approach, spurring students to think deeper and co-construct (Ben-Ari, 1998) concepts in a small group setting, through inquiry and application (Kussmaul, Mayfield, & Hu, 2017). We call our approach POGIL-like since POGIL is a copyrighted term to be used only by the POGIL project (*CS-POGIL | DCV (Directed, Convergent, Divergent) Questions*, n.d.).

POGIL-like is a pedagogy where students are organized into small teams. Students collaborate and actively learn to work together (Kussmaul, 2011). Students start each class session with little or no prior knowledge of the topic, so they can benefit from the co-construction of knowledge through POGIL-like activities without added misconceptions. The instructor serves as an active facilitator, walking around the classroom and helping students as needed during the session. Each team consists of 4-6 students, and each student has a specific role to play. There are 4 roles: Manager, Recorder, Presenter and Reflector. Students engage with "models" to learn the content and complete "activities" to exercise their understanding of the content. An overview of the POGIL-like pedagogy and its salient features can be found in our previous work (Gopal & Cooper, 2022).

More information on the types of questions, the POGIL-like "model" and activities can be found in our paper on POGIL-like (Gopal & Cooper, 2022). POGIL-like a combination of one or more models and a set of one or more activities in each session. Models contain content presented with tables, figures, action verbs, and some text. Each model is followed by one or more activities. A model-activity set combination is called a POGIL-like cycle. This cycle encourages students to explore (E), invent (I) and apply (A) the content concepts. To create the activities we used three types of questions -Directed (D), Convergent (C) and Divergent (V) questions (Gopal & Cooper, 2022). Several such POGIL-like learning cycles (E-I-A) consisting of models and corresponding D/C/V questions were used in our classroom implementation.

4.2. Study Context

Our research project was reviewed and classified as exempt by our University's Institutional Review Board. Students who chose to participate in the study did so by granting the researchers their informed consent. Data for this study were collected from 24 participants of a cohort of sophomore/junior students taking a software engineering class in the Fall of 2021. All students were taught using POGIL-like in the classroom in person.

4.3. Data Collection

Students were requested to answer the same online questionnaire at the beginning and end of the semester. We created this questionnaire specifically to target concept correlations between SE topics such as requirements gathering, Agile ceremonies and artifacts, and software testing. This open ended questionnaire asked students to describe what they thought of various topics and workflow in the real world. We used prompts such as "What is the role of requirements elicitation in SE?", "In agile software engineering, how does communication take place between team members and stakeholders?", and "When you get stuck during development, what would you typically do?" We combined student responses for these questionnaires with our real time field notes and reflective journal on students and their individual performances in various roles during the POGIL-like sessions (Emerson, Fretz, & Shaw, 2011). During our analysis, we corroborated students' written responses to the open ended questionnaire along with these field-notes and journal entries.

4.4. Data Analysis

Our data analysis consisted of content analysis (Hsieh & Shannon, 2005). We identified themes based on a mutimethod data collection (including participant/non-participant observations and field notes). Our primary goals during content analysis were to identify patterns in ideas as expressed by our students in their written responses. The first and second author of this paper coded our participants' written responses individually, generating and assigning over 230 codes. We employed frequency mapping of

codes iteratively both manually and using the HyperResearch software (*Qualitative Data Analysis tool - HyperResearch*, n.d.). We arrived at a code map (Anfara, Brown, & Mangione, 2002) which we used to help the process of code grouping and categorization into themes. We converged the code groups into the notes from our field notes and journals to come up with our themes by two separate coders independently and in parallel (Strauss & Corbin, 2015). In doing so, we were able to identify, verify and collate the themes we both noted.

We present the following data sections where we aim to forefront the opinions and perceptions of our participants in their own voices. We have followed existing research guidelines on how to position participant participation, and our method for meaning making involves understanding recurring expressions of participant sentiments and ideas by taking into account the context in which they were written and submitted (Creswell & Poth, 2018; Ketelhut & Schifter, 2011; Foley, 2002). In the following section, we highlight participants' voices by thematically presenting their written questionnaire responses. We use participants' own words as subheadings in each theme presentation. We used pseudonyms for our participants' actual names. Our primary focus in this study was to understand students' perceptions of SE topics. When instructed using the POGIL-like pedagogical approach, were students able to make connections between various SE concepts? How did their perceptions of SE concepts and topics relate to their identity in CS? These were the main questions we sought to answer.

4.5. Reliability

Intercoder agreement or inter-rater reliability is a measure of reliability commonly used in qualitative studies (Creswell & Poth, 2018). We focused on intercoder agreement to ensure the reliability of our coded data based on multiple coders interpreting the data through the process of coding, code mapping, categorization and theming. The first two authors of this paper, both trained in qualitative research methods, served as coders for our data. We calculated Cohen's Kappa (Hsu & Field, 2003) to measure intercoder reliability and we report an intercoder reliability (Creswell & Poth, 2018) of 0.9 (90%) among all themes that emerged from our 230 codes.

5. Results: Themes

In the following subsections we position the voices of our participants and start with how they broadly viewed the purpose of software. We explore their thoughts on requirements gathering, and the role of documentation. How do students think about communication in an agile SE environment, and how do they handle being stuck during everyday development?

5.1. Theme 1: Software is more than just a tool to solve business problems

We found that students viewed software as being important to end users, beyond just the business needs. While software did solve business problems, our students found it to be a source of potentially informing, educating, and ultimately, uniting end users.

"Software is often created with input from domain experts. They usually provide insights into how a problem would be solved without computers, and then developers use that solution to help them solve general cases." - Pete.

"Software is just more than to solve domain problems, more after this pandemic software use is increased exponentially. Software has the potential to unite people." - Kim.

"In my opinion, software can do more than just solving domain problems. It can also be used to inform, educate and help raise awareness on social issues. I think that software can be, and is, used everywhere in today's world. I think that's also obvious to see when every company hires a software engineer." - Tim.

"Software is used mostly for solving a company's specific problems. Software cannot be used to solve every single problem that a company may have, but if a company has a specific problem, software can be a great solution to solve it." - John.

"Software is used to solve problems. But it is also an experience for the user to enjoy. Sometimes (de-

pending on what type of software you are talking about) the purpose could be solely for enjoyment. Although the idea for software development is to solve a problem/addition that can be used by consumers." - Alia.

Software is mostly used to improve and solve modern problems efficiently, quickly, and accurately. Students disagreed with the thought that software is primarily used as a tool for solving problems specific to a business and its needs - they opined that it could do more than that. They expressed that while software could reduce the time it takes to solve or completely solve problems for a business need, it could also be used to enhance an end users experience of the solution. It seems that students struggled to articulate what exactly the additional value of software was besides solving business/domain problems.

5.2. Theme 2: The need for documentation varies based on the process model adopted

The role of documentation in SE takes center stage in this section. Our students went beyond the surface level understanding that documentation is required and helps maintain software; they understood and expressed that the amount and granularity of documentation required in a software development project depends on many factors, as we detail below.

"Documentation truly depends on what kind of approach you are taking to developing software. Although all approaches should have some documentation, the amount of documentation and when the documentation is created depends on the approach. For example, if a business is in need of high security software that does a pre determined set of functions and will not change as it is vital software, a business will probably take a waterfall approach...documentation is followed and developed depends on the needs of the customer and the kind of software that is being developed." - Matt.

"Documentation will depend largely on the business's needs and what type of software project it is. If the business knows exactly what they need up front, and give those needs to the development team, documentation is more important. If a project is made agilely, the documentation may be more minimal because the software is everchanging." - Paul.

"There are times when documentation is not needed. One scenario would be when you are making software for yourself, and there really is not any reason for documentation to slow down the process." - Linda.

"Depends on the type of business needs driving the software, because I don't think it's so clean cut. For some businesses that just need simple software like the addition of a search bar to their online store doesn't need too much, it can be minimal. However a place like a hospital dealing with things like patient records should be pivotal because of all the care that needs to be taken with a database and formatting of records along with other requirements that need to be in place." - Lupe.

"It is absolutely critical that people other than you can understand your code. Even if you step away from your code, it makes it easier for you to also understand your code. In a waterfall method, there isn't a ton of revolving documentations, whereas in agile, the documentation is necessary." - Hannah.

Our students displayed a strong understanding of the idea that documentation is important, but depends on the business needs. They expressed an understanding of the different documentation approaches warranted by different software process models. They understood that how important documentation is can depend on a variety of factors, and that for some businesses it can be critical, but for some others it can be relatively minimal or unnecessary.

5.3. Theme 3: Timely and frequent communication between team members and stakeholders is essential

We can classify our participants' voices on agile communication under three topics: frequency of communication, parties involved in communication and the importance of communication. Our students displayed a good understanding of the evolving nature of requirements (when developing software using an agile process) and how the frequency of communication within the team and with stakeholders could affect the quality of the end product.

"Requirements change in every sprint - sometimes they could change drastically. It is very important that we keep communicating with the stakeholders and the other devs so that we capture and code to the correct requirements for each sprint." - Matt.

"Scrum masters, project managers, developers and product owners should all be involved in regular communication. Daily scrums for the development team, and at least weekly meetings with the stakeholders are crucial." - Sarah.

"The quality of the software being developed is highly dependent on how well the team communicated with itself and with the clients. Without getting requirements right, code won't be done correctly, and tests will be testing bad code that doesn't reflect what the customer wants." - Rachel.

"Without constant communication, the product will fail - how will developers know what the clients want as requirements change? How will they develop and test?" - Alia.

Based on the above statements, we can see that our students understood and advocated frequent communication between all parties involved - daily for team members and weekly for other stakeholders. Students also clearly understood the evolving nature of requirements on an agile project, and the importance of timely communication with stakeholders so that the correct requirements were captured for each sprint. The importance of meaningful and effective communication cannot be overstated in an agile environment, and our students exhibited an understanding of the ramifications of ineffective communication.

5.4. Theme 4: Handling downtime and time management in software engineering

We asked students how they would proceed when they were unable to proceed during development due to limiting circumstances. They had to put themselves in the context of a professional software engineering job, and had to think through the lens of a junior software engineer. The following comments illuminate their thought process on this topic. This is important to know because how a student proceeds when they are stuck, could indicate how they perceive themselves in the context of software engineering.

"Take a break, even though we are just sitting down coding drains a lot of energy, generally a lot of programmer feel mental stress, so relaxation is really important. I would take a time out and relax bring my way back to work with stable mental health." - Tim.

"During downtime I would want to try and relax to clear my mind so I could think of possible projects to work on that would be beneficial to my role at the company. I would also want to mingle with other developers to learn more about different types of projects." - Asher.

"I would probably work on resolving tasks in the sprint backlog. If there isn't anything in the sprint backlog, I guess I would create some more unit tests just to make sure the methods work. If there are already plenty of unit tests, I suppose I would work on some documentation or double checking the code for code smells." - Mike.

"Continue working on other items, bugs, or features that needed to be done. Interact with co-workers and gain insight with what they are working on or take a longer break. There is always a cycle to it and eventually you need to take advantage of it." - Wayne.

"When I have downtime during a workday as an industry professional I would try to pick up more responsibilities from the team and try to learn new skills." - Cody.

There are several threads that emerge from the above quotes: students want to individually help move the sprint/product backlog forward, focusing on immediate development needs; students wish to help their fellow teammates, fostering a sense of solidarity and teamwork; students want to improve their technical skills - learning new technologies and methodology elements; and finally, students recognize the mental stress that comes with agile software development with its constant deadlines, and wish to use downtime to focus on improving their mental health.

6. Discussion

6.1. Key takeaways

The most important takeaway from our analysis is that our students connected elements of several software engineering topics together. Their understanding of many topics was practical, well informed and deeper than a simple surface level understanding of said topic. For example, they were able to relate requirements to testing, communication to requirements, and the Agile process to end user focused software design. This is noteworthy because this was the first SE course that most of our students had encountered in their academic preparation. Students also expressed a clear sense of belonging in the way they spoke about SE topics, and their opinions on various issues relating to them.

Several interesting themes emerge from our analysis. First, our students said that software is more than a tool to simply solve domain problems. They perceived software development in general as leading to a product for the end user to enjoy. Our students also saw software as a living and breathing product. Changes could be constantly made and updates could be pushed out every other sprint. Software can therefore adapt to changes in the real world, in terms of business needs and end user human needs. It is interesting to note how students captured the intangible idea of software being more than the sum of its parts, with an ability to potentially inform, educate, and/or uplift human beings. In summary, according to our participants, software is designed to solve domain problems, but can be much more. The "much more" primarily related to other people using the software, and their experience and level of enjoyment with it. However, this idea of UX being separate from the actual software is indicative of an incomplete understanding of some aspects of SE.

Second, our students exhibited a mature understanding of the role of documentation in software development, recognizing that documentation, in most cases, will depend largely on the business's needs and what type of software project it is being developed. We see again that there is a sense of community and a recognition of their role as a software engineer being tied to the larger group of engineers in a company or enterprise.

Third, our students saw that requirements change in an agile environment, and requirements engineering is a tool for effective communication to help clear doubts and misunderstandings among developer and stakeholders. Students expressed that the role of requirements elicitation was to come to an agreement between the consumer (stakeholder) and development team of what the exact parameters of the software will be. Again, we see a recognition of software being engineered in the context of several people-stakeholders, end users, and development teams.

Next, our students understood the benefits of effective communication between all parties involved - development team, product manager, project manager, stakeholders, end users, and management. They also understood that when effective and continued communication is broken, the product suffers. Students also recognized that software quality depended heavily on reliable and steady communication between parties.

Finally, when asked how students would handle downtime, they almost unanimously expressed a wish to help other teammates, or take on more work from the existing sprint/product backlog. Some students also mentioned the importance of taking time to care for their mental health, recognizing the stress and mental strain that real world software development can entail.

Our analysis reveals that our students prioritized business needs, were curious, eager to take on more work as needed, with a strong work ethic, and strove hard to maintain good communication and transparency with their stakeholders and development team. These are in agreement with our previous study (Gopal, Cooper, & Bockmon, 2021) where we heard from industry partners on the advice they would give SE students to succeed in the industry, based on their interactions with and observations of students in a peer instruction (Mazur, 1997) based SE course. By varying the pedagogical approach to POGIL-like but keeping the content and instructor unchanged, and with similar prior academic preparation, our participants seemed to have gained some other things: Possibly due to the sustained collaboration and concept invention aspects of POGIL-like, our students developed a strong sense of where they belonged

in a software development team, in relation to stakeholders, and end users. They also got to apply their newly invented concepts in each POGIL-like session (E-I-A cycle) and had a more realistic grasp on the exhilarating and potentially demanding nature of individual and collective software development.

6.2. POGIL-like: its influence on student perceptions and student motivation

Literature shows that autonomy and self-driven inquiry has been showed to increase student motivation (Buchanan, Harlan, Bruce, & Edwards, 2016; Biggers et al., 2008). Student motivation is linked to the student perceived value or meaning in the academic work at hand. Student interest increases cognitive and affective outcomes, specifically student motivation (Ainley, Hidi, & Berndorff, 2002). "Student control of the learning process," not only influences academic achievement, but greatly increases student motivation (Mega, Ronconi, & De Beni, 2014).

Perhaps the largest difference in instructing students using traditional lecture vs POGIL-like is the autonomy afforded to students. To enable problem solving, critical thinking and reasoning skills, the D/C/V question pattern in POGIL-like creates fertile ground for inquiry based discovery: initial direction (D), convergence of knowledge from the directed discoveries (C) leading to co-construction and invention of concepts, and finally, divergent application of the newly constructed concepts in unfamiliar scenarios. In a topic like DevOps, for example, with a specific concept like continuous integration, there could be several ways of approaching the topic. With traditional lecture, the topic maybe taught with slides that all students receive. With POGIL-like, with the overarching guidance of the activities (with several D/C/V questions in E-I-A cycles) and models, each individual student explores the topic on their own. Each student invents the concept together with their team mates, co-constructing the ideas that thread together the complete concept, with the help of the model and activities. There is a rationale for every step in these activities, and students have to think about the "Why" in addition to the "How". Students have to then individually apply the newly invented concept - and this can be different for each student as well. We see that there is a high level of autonomy in each step of the E-I-A cycle, and a high level of engagement is required to complete each activity. The high level of autonomy in the POGIL-like paradigm allows students ways to think and function independently, as well as contribute freely within a group.

7. Threats to validity

Lincoln and Guba (Lincoln & Guba, 1985) indicate that validity in qualitative studies is expressed as respondent/participant bias, reactivity, and researcher bias. As for participant bias, there is always a possibility that our students responded to the questions based on what they thought was the right or acceptable answer instead of what they really felt. We took care to explain to participants that they received participation credit, not correctness credit. We also ensured that students knew that there were no right or wrong answers to the survey questions. The primary author of this paper acknowledges that her vast experience and background in the software engineering industry is likely to have influenced her interpretation of the the data with a marked bent towards industry relevant information. We attempted to lessen the effects of these aspects (Robson, 2002), we triangulated student response data with our own reflections and journals and audits.

Data saturation was achieved based on the guidelines by Creswell and Poth (Creswell & Poth, 2018). There is always a possibility that in spite of our systematic thematic analysis, other researchers may infer and construct different themes from our raw data. We are confident in our findings to be valuable and relevant within the context of our study because we triangulated our data from 24 student responses from a single cohort with the same instructor and instructional pedagogy with the same topics of instruction.

8. Conclusion and future work

In answering our research question, "Were undergraduate students able to learn from and build upon multiple relevant concepts to display a connected understanding of software engineering topics when instructed using a POGIL-like pedagogy?", we conclude that our students did indeed connect various software engineering topics in constructing their understanding, and displayed synthesis skills, nuance

and depth in explaining their impressions, when instructed using our POGIL-like pedagogy. Students showed analysis and synthesis skills beyond just basic knowledge where they could tie together topics like requirements engineering and testing, or communication with all aspects of software engineering. This leads us to believe that students were able to approach the higher layers of Bloom's taxonomy during their learning (Bloom, 1956). We think that this deeper understanding was fostered by the structured, process oriented approach in POGIL-like, specifically with the collaborative learning cycles utilizing the various types of Directed, Convergent and Divergent (D/C/V) questions the Explore-Invent-Apply (E-I-A cycle).

Through the structured process oriented POGIL-like approach, which places a heavy emphasis on collaboration and co-construction of knowledge, we find that students displayed a strong and connected understanding of the agile software development process and were able to place themselves in it. Their sense of identity within SE was revealed by their perceptions of SE topics, and we find that the inquiry based, constructivist learning approach through POGIL-like helped students relate themselves within the individual software developer context as well as the larger software engineering context involving the rest of the development team, product managers, project managers, stakeholders and end users. Previous literature shows that a positive self perception of one's ability was a key trait that helped retain students (Almulla, 2020).

POGIL-like assigns students into four distinct roles- manager, presenter, recorder and reflector. Our future work directions include understanding the impact of these specific POGIL-like roles in building students' confidence and motivation in SE, and whether other constructivist or collaborative pedagogies have similar effects on student affect. We also intend to compare our findings with a traditional lecture class with the same content and instructor, and delve deeper into how students' impressions varied between both approaches.

9. Acknowledgements

We would like to acknowledge and thank Dr. Justin Olmanson, Associate Professor, College of Education and Human Sciences at the University of Nebraska-Lincoln for his generous guidance on qualitative analysis techniques.

10. References

Agosto, D. E., Gasson, S., & Atwood, M. (2008). Changing mental models of the it professions: A theoretical framework. *Journal of Information Technology Education: Research*, *7*(1), 205–221.

Ainley, M., Hidi, S., & Berndorff, D. (2002). Interest, learning, and the psychological processes that mediate their relationship. *Journal of Educational Psychology*, *94*(3), 545.

Almulla, M. A. (2020). The effectiveness of the project-based learning (PBL) approach as a way to engage students in learning. *Sage Open*, *10*(3), 2158244020938702.

Ames, C., & Archer, J. (1988). Achievement goals in the classroom: Students' learning strategies and motivation processes. *Journal of Educational Psychology*, *80*(3), 260.

Anfara, V., Brown, K., & Mangione, T. (2002). Qualitative analysis on stage: Making the research process more public. *Educational Researcher*, *31*(7), 28–38.

Bean, J. P., Eaton, S. B., et al. (2000). A psychological model of college student retention. *Reworking the student departure puzzle*, *1*, 48–61.

Ben-Ari, M. (1998). Constructivism in computer science education. In (Vol. 30, pp. 257–261). ACM New York, NY, USA.

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin*, *40*(1), 402–406.

Bloom, B. (1956). *Taxonomy of educational objectives, handbook 1: Cognitive domain* (2nd edition Edition edition ed.). New York, NY, USA: Addison-Wesley Longman Ltd.

Buchanan, S. M. C., Harlan, M. A., Bruce, C., & Edwards, S. (2016). Inquiry based learning models, information literacy, and student engagement: A literature review. *School Libraries Worldwide*, *22*(2), 23–39.

Creswell, J., & Poth, C. (2018). *Qualitative inquiry and research design: Choosing among five approaches* (4th ed.). Sage Publications, CA.

CS-POGIL | DCV (Directed, Convergent, Divergent) Questions. (n.d.). Retrieved 2021-12-30, from https://csPOGIL.org/DCV+(Directed,+Convergent,+Divergent)+Questions

Dempsey, J., Snodgrass, R. T., Kishi, I., & Titcomb, A. (2015). The emerging role of self-perception in student intentions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 108–113).

Emerson, R. M., Fretz, R. I., & Shaw, L. L. (2011). *Writing ethnographic fieldnotes*. University of Chicago Press.

Foley, D. E. (2002). Critical ethnography: The reflexive turn. *International Journal of Qualitative Studies in Education*, *15*(4), 469–490.

Gopal, B., & Cooper, S. (2022). POGIL-like Learning in Undergraduate Software Testing and DevOps - A Pilot Study. In *Proceedings of the 27th Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (p. Accepted).

Gopal, B., Cooper, S., & Bockmon, R. (2021). Industry partners' reflections on undergraduate software engineering students: An exploratory pilot qualitative study. In *Proceedings of the 32nd Annual Psychology of Programming Interest Group Workshop(PPIG).*

Hsieh, H.-F., & Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research*, *15*(9), 1277–1288.

Hsu, L. M., & Field, R. (2003). Interrater agreement measures: Comments on Kappan, Cohen's Kappa, Scott's π, and Aickin's α. *Understanding Statistics*, *2*(3), 205–219.

Ketelhut, D. J., & Schifter, C. C. (2011). Teachers and game-based learning: Improving understanding of how to increase efficacy of adoption. *Computers & Education*, *56*(2), 539–546.

Kussmaul, C. (2011). Process oriented guided inquiry learning for soft computing. In *International Conference on Advances in Computing and Communications* (pp. 533–542).

Kussmaul, C., Mayfield, C., & Hu, H. (2017). Process oriented guided inquiry learning in computer science: The cs-pogil & introcs-pogil projects. In *ASEE Annual Conference and Exposition, Conference Proceedings* (pp. 1–7).

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. Sage Publications, CA.

Mazur, E. (1997). *Peer instruction a user's manual*. Prentice Hall.

Mega, C., Ronconi, L., & De Beni, R. (2014). What makes a good student? how emotions, self-regulated learning, and motivation contribute to academic achievement. *Journal of Educational Psychology*, *106*(1), 121.

Melnik, G., & Maurer, F. (2005, 06). A cross-program investigation of students' perceptions of agile methods. *Proceedings - 27th International Conference on Software Engineering, ICSE05*, 481-488.

Peters, A.-K., & Pears, A. (2013). Engagement in computer science and it–what! a matter of identity? In *2013 Learning and Teaching in Computing and Engineering* (pp. 114–121).

Qualitative Data Analysis tool - HyperResearch. (n.d.). Retrieved from http://www.researchware.com/products/hyperresearch.html (Last Accessed March 2021.)

Robson, C. (2002). *Real world research: A resource for social scientists and practitioner-researchers*. Wiley-Blackwell.

Souza, M., Moreira, R., & Figueiredo, E. (2019). Students perception on the use of project-based learning in software engineering education. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering* (pp. 537–546).

Strauss, A., & Corbin, J. (2015). *Basics of qualitative research: techniques and procedures for developing*. Sage Publications, CA.

Tinto, V. (1997). Classrooms as communities: Exploring the educational character of student persistence. *The Journal of Higher Education*, *68*(6), 599–623.