# Dimensions of Experientialism for Software Engineering Education

Reid Holmes
University of British Columbia
Vancouver, BC, Canada
rtholmes@cs.ubc.ca

Meghan Allen
University of British Columbia
Vancouver, BC, Canada
meghana@cs.ubc.ca

Michelle Craig
University of Toronto
Toronto, ON, Canada
mcraig@cs.toronto.edu

## ABSTRACT

There is a gap between the abstract concepts taught in the classroom and the skills needed for students to succeed once they join the workplace. The Undergraduate Capstone Open Source Projects (UCOSP) program was developed to narrow this gap by enabling undergraduate computer science students to have an experiential software engineering learning opportunity. Over the past 8 years, 737 students from 30 universities have taken part in this program.

In this paper, we sought to understand student perceptions of how UCOSP complements traditional classwork by providing real-world software engineering exposure. We report on a qualitative analysis of 2,203 quotes collected from 167 students from 18 universities over six academic terms. We analyzed these data using a grounded theory approach based on open coding to gain insight into the key benefits of the program from the students' perspective. We found that students highly value being able to apply their classroom knowledge to real, novel tasks, for real projects with a community of users, while receiving real mentorship from a member of the development team. Further, we found that contributing to real software systems provides greater understanding of software engineering than might otherwise be obtained through more traditional means.

Our goal is that our analysis can help fellow educators add additional experimentalism into their existing programs.

## CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → *Open source model*;

## KEYWORDS

experiential learning, software engineering education, capstone

**ACM Reference Format:**
Reid Holmes, Meghan Allen, and Michelle Craig. 2018. Dimensions of Experientialism for Software Engineering Education. In *Proceedings of 40th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET'18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3183377.3183380


*ICSE-SEET'18, May 27-June 3, 2018, Gothenburg, Sweden*

ACM ISBN 978-1-4503-5660-2/18/05…$15.00
https://doi.org/10.1145/3183377.3183380

## 1 INTRODUCTION

Experiential education has been proposed as a way for learning *in situ* enabling students to have non-traditional educational experiences that link work, education, and personal development [14].

The Undergraduate Capstone Open Source Projects (UCOSP) program was created to provide students with a course-based educational experience that, as much as possible, parallels software engineering in the real world [24]. Students are matched with open source projects, work on distributed teams, and are mentored by professional members of the development team. This program has been running for 8.5 years with 30 post-secondary institutions and a total of 737 students.

For the past five years we have instrumented UCOSP to collect data about its efficacy. We deployed three surveys each term to investigate students' activities in the program and their perceptions of the program. One hundred and sixty-seven students who participated in the program, from 18 different Canadian universities, consented to participate in at least one of the surveys. We performed a high-level qualitative analysis of this survey data using a grounded theory approach based on open coding and found that students' experiences in UCOSP are overwhelmingly positive. From these data, three high-level aspects of experientialism emerged. First, students appreciated working on real projects. Second, they found value in performing real tasks they viewed as having practical impact. Finally, they valued receiving real industrial mentorship.

These three dimensions of reality provide a lens to examine the experientialism of an intervention as it applies to software engineering education. First, students can be exposed to *real projects*. These projects expose students to software systems that are used in practice and have actual users who can be impacted by their contributions. Second, students can be given *real tasks*. That is, the tasks they perform for the projects provide novel, valuable functionality. Finally, students can be guided through the software engineering process by *real mentors*. That is, the mentors originate in the projects themselves. These mentors are able to provide effective guidance to the students in how software is built in practice, and are able to shepherd contributions so they can be deployed.

The main contribution of this paper is a qualitative analysis of student perceptions of a experiential software engineering education program collected via a longitudinal study. From this analysis we demonstrate that:

- Working on real projects helps students understand the complexity, tools, and processes used to build production systems.
- Students value working on real tasks that they perceive as having impact on real end users.

- Industrial mentorship provides a wealth of tangential benefits to software engineering capstone courses.

We hope our analysis can help curriculum-design committees consider these aspects of experientialism when augmenting their programs with more experiential-oriented courses and activities.

## 2 BACKGROUND AND RELATED WORK

Employers, educators, and researchers have long recognized a conceptual gap between what computer science students learn in a typical undergraduate education, and the skills they need as they begin careers as software developers [4, 9, 20]. One recurring theme is a push for educators to provide students with the opportunity to work with current industry tools and processes on legacy codebases with long histories and communities of users and already-existing technical debt [1, 8, 16]. In other words, for students to work on *real projects*.

### 2.1 Real Projects

Some software engineering educators have addressed the real projects dimension of experientialism by using existing open-source (OSS) projects [1, 8, 10, 11, 15, 16] with large existing codebases. Often this codebase is only used as a starting point for the student projects and while students make improvements to the code – either fixing bugs or adding new features – the changes don't ever become part of the official release of the project [5, 16]. Allen et al., require students to make improvements to a project (Dr. Java) owned by their department [1]. Billingsley and Steel try to mimic a real development community by having groups of 3-4 students each add different features to the same OSS project [5]. Students are gaining useful experience with existing projects in these courses, but in UCOSP each student also makes a contribution that will be deployed.

Other instructors mimic real projects with course-specific, non open-source projects. Szabo had students work on existing codebases from student projects from a previous offering of the course [25], while Murphy et al. had students work on codebases from a different course at the same institution [17]. Nurkkala and Brandle developed a software studio model where student teams work on one project over multiple years [19].

All of these courses meet our first dimension of experientialism, real projects, as they had existing codebases, but only Fagerholm et al. integrated real mentors and real tasks [10, 11]. In other words, some aspects of the learning experience in most of these courses were simulated rather than real.

### 2.2 Real Users and Clients

Some courses focus on the real tasks dimension of experiential learning by designing courses where the projects (often not existing before the onset of the course) are developed to meet a genuine need of actual users [1–3, 6, 7, 17, 18, 21]. In these courses, students are motivated to produce a usable final product, but they do not experience the challenge of working with legacy code and are not forced to work with existing processes and tools. Students are also are not a part of a large development community with real mentors.

Of the courses that work with real clients, some require all students to complete novel functionality [3, 6, 7, 19, 21], thus meeting our definition of real tasks. Others require students to complete real tasks proposed by students rather than a client [5] or from an open source project owned by the department [1]. Ellis et al. describe multiple ways in which students can make novel contributions to HFOSS projects ranging from updating documentation to contributing code [8]. These course projects provide more experiential software engineering exposure than traditional courses in which every student works on the same project, but in some cases the students are writing greenfield code or are unmentored, and therefore not following industrial process or tools.

### 2.3 Real Mentors

In comparison to a typical software engineering course, where the entire class solves the same assignments, a real-world course where each group is writing code for a different feature or a different project is much more work to manage [22]. Many courses hire project managers, coaches, or mentors to alleviate the mentorship burden from the instructor [3, 7, 17, 18]. Very few courses provide real mentorship via existing developers from the project team. Allen et al. provide mentorship to the students by hiring the existing project developers as teaching assistants to each manage a group of 2-6 students working on a different new feature [1]. Bloomfield et al. recruit local software developers to mentor their students as they create software for a non-profit, but these developers are not from the non-profit [6].

To our knowledge, the only other software engineering educational program that provided real projects, real mentors, and real tasks was the Facebook Open Academy, a collaboration between Stanford and Facebook from 2012–2014 in which students from 25 international universities worked on 22 open source projects and were mentored by members of the open source team [10].[1] Fagerholm et al. argue that the mentoring process used by Facebook Open Academy and UCOSP is not only an effective method of experiential learning for students but also a useful technique on-boarding new open source developers in general [10].

### 2.4 Undergraduate Capstone Open Source Projects

Like other programs, the Undergraduate Capstone Open Source Projects (UCOSP) capitalizes on the availability of open source projects with established developer communities and existing users. But rather than simply using these projects as a source of legacy code and simulating the rest of the experience, as was done by McCartney et al. [16] and Billingsley and Steel [5], UCOSP provides all three aspects of experientialism.

Each semester, UCOSP recruits open source projects to participate in the program. Some projects have participated consistently for many terms. Our expectations are that the projects are pre-existing, have a developer community, and have a user base. As one example, ReviewBoard has consistently been one of the project participants. ReviewBoard is an open source code review tool that is used by thousands of open source and industrial teams [13]. The ReviewBoard codebase has approximately 600,000 lines of code and 8,574 commits from 181 contributors.

[1] In 2013 the UCOSP code sprint was co-located with the Open Academy Hackathon and in one semester of 2014, UCOSP fully participated in the Open Academy.

| ID | # Cards | Question Text |
|---|---|---|
| **Q1** | 348 | What do you feel you gained from your UCOSP experience? |
| **Q2** | 345 | Was UCOSP any different than your prior courses? If so, how? |
| **Q3** | 282 | Did UCOSP motivate you differently compared to in-class courses? |
| **Q4** | 279 | What specific skills from previous CS courses prepared you for UCOSP? |
| **Q5** | 243 | How do you keep up with the current project status? |
| **Q6** | 242 | What aspects of the course were most useful? |
| **Q7** | 237 | What aspects of the course were a waste of time? |
| **Q8** | 327 | If you have had a co-op term or paid internship, did you learn anything different from your UCOSP experience? |

**Table 1: Post-program survey questions. The answers to these questions were split into 2,203 cards.**

Students apply to UCOSP and receive course credit – typically an upper-year project or directed-studies credit – via their own institutions.

A faculty member at each institution commits to selecting and supervising their students who participate in the project. These home faculty members meet with the students biweekly to ensure that they are making acceptable progress. Due to financial and logistical constraints, each institution may only select between two to six students. Demand for UCOSP is often high and therefore only the strongest students are selected. The supervision required by the home institution faculty member is light in comparison to the supervision that would be required in a typical upper-year project or directed-studies course.

The UCOSP program matches students with the open source projects based on the students' preferences and team sizes that are set by the project mentors. Team sizes typically range from three to 10 and include students from multiple institutions.

The course kicks off with a three day, face-to-face sprint which is mandatory for all students and project mentors. Once the sprint is over, students continue to work on their tasks and typically communicate with their project mentors and teammates via the usual channel for their project. They often have online video meetings weekly in addition to using chat tools and issue tracking systems.

A steering committee, currently comprised of four computer science faculty members from Canadian institutions, oversees the project selection, student/project matching, sprint organization, and grades. The steering committee requires project mentors and home institution faculty supervisors to agree to meet the program's expectations for meeting with students, submitting feedback, and assigning grades. Project mentors are required to give mid-term and end-of-term feedback that is relayed to the students. The steering committee collects suggested final grades from the mentors, vets them for consistency, and shares them with the home institution faculty supervisors who assign course grades.

For further information on UCOSP, Stroulia et al. [24] provide a thorough description of the program and Holmes et al. [12] discuss the logistics of the program and the lessons that the steering committee learned over the years.

## 3 METHOD

The objective of this research project was to understand the benefits students felt they received from UCOSP, specifically in terms of how these benefits differed from their traditional coursework. Specifically, we wanted to answer the following research question:

**RQ** How do experiential software development projects augment traditional computer science curricula?

To answer our research question, we performed a longitudinal qualitative evaluation that surveyed students who participated in UCOSP to gain an understanding of their perception of the program and how it related to their normal coursework.

### 3.1 Participants

For our evaluation we surveyed 167 students. To gain admission into the UCOSP program these students had to be in third or fourth year and be selected by faculty at their home universities. In general, faculty were advised to select their strongest students who they believed would benefit from the UCOSP program. Universities that had strong co-op programs often tried to select participants who did not have co-op experience. The 167 survey respondents came from 18 different universities across Canada. These schools ranged from small local universities with $< 3{,}000$ students, to large research-intensive universities with $> 60{,}000$ students. Our participants were surveyed over six different academic terms between September 2013 and April 2016.

### 3.2 Data

We collected feedback from students at three specific points:

**Pre-program:** After students were selected by their home university faculty, but before they had been assigned to projects, we asked them several questions about their project preferences and past experiences.

**Post-sprint:** All students in the program met together for a co-located sprint over a single weekend. After this was complete we solicited feedback on their sprint experience.

**Post-program:** Once the program was complete, but before they had been given their feedback and assessment, we conducted a final survey to gather feedback on the program.

Throughout the informed consent process students were clearly told that the UCOSP steering member conducting the surveys would have no access to the grades, and those involved in assessment would only see anonymized survey data (and even then only after the term was over and grades were submitted).

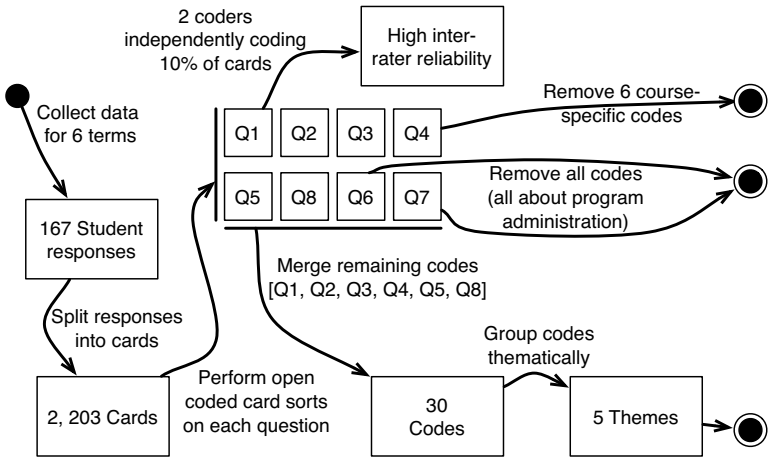


**Figure 1: Qualitative analysis process.**

Students were required to complete the surveys at the conclusion of the course. Approximately 95% of students consented to have their responses included in this study.[2]

After a cursory examination of the data, we concluded that the pre-program and post-sprint surveys were mainly of interest for the administration of the program. We also realized that the post-program surveys best reflected the students' experiences and provided the most insight into the value of the program. The remainder of this paper only focuses on these post-program questionnaires; the questions asked of the students are given in Table 1.

### 3.3 Analysis

We analyzed these data using an grounded theory approach based on open coding [23]; an overview of our coding process is given in Figure 1. We ran an automated program to transform each student response into a series of cards. Each card consisted of a single sentence; sentences that only consisted of a single word (usually "Yes." or "No." were joined to the sentence that followed it). After this process, we had 2,203 total cards.

All three authors worked to sort the cards for each question independently to derive question-specific codes. Each card was assigned a single code, but if tricky cards were encountered we would discuss the best code to assign them to. After coding each question we examined all the codes for cohesiveness and merged smaller codes and refined larger codes as needed.

To ensure we were able to consistently code our cards, we performed an inter-rater reliability analysis. Two coders independently coded the same 40 of the 348 cards from Q1. We used Krippendorf's alpha to capture our agreement for this coding exercise; the resulting score of 0.87 indicated a high level of agreement between the two coders.

Once each question had been processed independently, we next analyzed the codes between questions to see how they could be merged into a comprehensive overview of the students' feedback. While doing this we realized that six codes from Q4 were related to previous courses that the students had taken and so we subsequently removed them from the rest of the analysis. We also found that Q6 and Q7 were mostly about program administration; these codes were not merged with the other categories, although the cards themselves were not removed. The remaining codes from Q1, Q2, Q3, Q4 (less the six removed codes), Q5, and Q8 were then merged and given consistent names. During this process, five high-level themes were also identified to provide further hierarchical context to the individual codes.

[2]The exact figure is not available; the responses from non-consenting students were deleted as a part of the anonymization process and their original records destroyed.

## 4 RESULTS

In this section we present the results of the qualitative study. The full list of codes from the study can be seen in Table 2. The 'general' code within each theme corresponds to quotes which fall within the theme but not one of the more specific codes. To summarize each code, we created a 'synthetic quote' to capture the kind of sentiment reflected by the cards sorted into that code.

We will first discuss the first three of the five high-level themes (Real Projects, Real Tasks, and Real Mentorship) as these provide the most interesting insight into the program from the students' perspective. We then discuss the final two themes (Soft Skills and Technical Skills) as they are somewhat less surprising given the project-based nature of UCOSP.

Students were almost universally positive about their UCOSP experience, "*the whole UCOSP journey was very rewarding and a great learning experience*" *(Q7. 2014.04_P05)* and that the time was well spent, "*There was no wasted time and every hour I spent on the project felt useful*" *(Q7. 2014.04_P24).*

### 4.1 Real Projects

By its nature, UCOSP forces students to work on legacy codebases that are deployed to real users. Students **appreciated the challenge** these large projects provided:

"*I learned how to deep-dive into a large codebase, trace how things*

| Theme | Topic | Synthetic Quote |
|---|---|---|
| Real Projects | General | |
| | Large systems & codebases | I worked with a large real system. |
| | Communication processes | I learned how to communicate in a distributed setting. |
| | Communication tools | I used tools to communicate in a distributed setting. |
| | OSS process | I learned how to get started with and contribute to an OSS project. |
| | OSS projects | I worked on an OSS project. |
| Real Tasks | General | |
| | Impact on real users | My work had meaningful impact on real users. |
| | Real requirements | The project was not a throwaway toy and had novel requirements. |
| | Code quality | I experienced the importance of quality & testing in real systems. |
| | Code review | I experienced code review. |
| | Applied learning | My work was applied and was not just 'in theory' like regular coursework. |
| Real Mentorship | General | |
| | Mentorship | My mentor provided meaningful feedback, tasks, and contacts. |
| | Professional developers | I worked with professional / seasoned developers. |
| | Professionalism | UCOSP feels more like work/co-op than a course. |
| | Networking | UCOSP will help me build my professional network. |
| | Independent learning | I developed techniques for learning how to make progress independently. |
| | Distributed teamwork | I learned how to work with a remote/distributed team. |
| Soft Skills | General | |
| | Resume building | UCOSP will help me fill a gap in my resume. |
| | Career planning | UCOSP will help me think about my future carrer direction. |
| | Communication | I improved my communication skills as a software developer. |
| | Teamwork & collaboration | I learned how to work in a team. |
| | Time management | I learned how to manage my time, meet deadlines, and motivate myself. |
| Technical Skills | General | |
| | Languages & frameworks | I was exposed to new/relevant languages and frameworks. |
| | Tools | I was able to use new/relevant development tools. |
| | Program understanding | I learned to read and work with other people's code. |
| | Programming experience | UCOSP provided me with practical development experience. |

**Table 2: Overview of high-level themes and subtopics within those themes from the open coded card sort. Each theme had a *general* topic to capture quotes from within that theme that did not otherwise fit in a more specific topic.**

*worked, and figure out how to accomplish what needed to be done.*" *(Q1. 2015.12_P17)*

The idea that these projects also originated in industry provided additional motivation, "*complex code base with a major company.*" *(Q1. 2014.04_P25)* One major challenge in this environment was **complexity of the systems** students were forced to learn:

"*At first it was very intimidating to jump into such a massive project but I eventually learned how to effectively navigate through a project of this scale, to take the extra 5 or 10 mins to ensure a method you're thinking about writing hasn't already been implemented somewhere, and how to manage and switch between different branches associated with separate tasks.*" *(Q1. 2015.12_P25)*

Several students referenced the differences between the **greenfield development** that they would typically experience in their course work and working on an established project:

"*The fact that some of the existing code wasn't that great was actually a plus as it gave me more practice identifying problems and making sense out of other people's decisions.*" *(Q1. 2015.12_P12)*

These projects really forced students to **work independently** to effectively contribute, especially since they were making novel contributions to the projects (in contrast to writing the same code as their classmates), "*Also UCOSP promotes self-learning and trying to figure out answers to your questions yourself before asking the mentors.*" *(Q6. 2014.12_P35)* These sentiments also appeared within the Technical Skills theme as students had to think about different ways to understand their systems.

By contributing to **active development communities**, students were forced to adopt their tools, processes, and cultural norms:

"*The most important thing I have learned is how to get started on contributing to an open source project, including (but not limited to) how to start working on an existing codebase, how to communicate*

*with the community to get help (and later on how to help others).*" *(Q1. 2014.12_P29)*

In particular, the community aspect of the project worked to encourage students to behave appropriately, "*It was an amazing experience to see to see the dynamics of an open-source project and get to participate in it.*" *(Q1. 2014.04_P17)*

Working with these systems also emphasized the **importance of real development processes** on overall success:

"*This gave me an idea of what real software development progress can be like and I learned about different techniques for documentation, continuous integration, and automated testing.*" *((Q1. 2015.12_P02))*

These processes were also **different than in the classroom** due to the different processes used by each project "*Learning new processes and systems that are not taught in school.*" *(Q6. 2014.12_P02)*

Many students also expressed positive feelings about **contributing to OSS systems**:

"*UCOSP was a great introduction to being an open source contributor for me, so much so that in addition to contributing I will be a mentor for new batches of students starting this summer.*" *(Q1. 2014.04_P09)*

By working on real projects students were subjected to many of the difficulties facing professional developers. This was meaningful in terms of the scale of the code they were working with and the complexity of the interactions within the codebases. This forced students to reason differently about the code they were working with since all of it was new to them and written by other developers.

## 4.2 Real Tasks

The tasks that students work on are expected to be novel contributions to their projects. This means the work must be of value, must end up being deployed as part of the project, and must not be trivial. Interestingly, one aspect of these tasks students most appreciated was their **impact on real users**:

"*In some other courses it can sometimes feel like you're creating an application just for the sake of doing it, where as almost everything we wrote for UCOSP is be used by thousands of people every day.*" *(Q2. 2015.12_P25)*

These sentiments were echoed by many students: "*make a contribution to a great product with thousands of real users*" *(Q1. 2014.12_P21)*, "*actual software used by thousands of users*" *(Q1. 2015.12_P15)*, "*real world project that solves real world problems.*" *(Q2. 2014.12_P29)*, and

"*We have had a few non-UCOSP people comment on our projects – confirming that other people are indeed using our project. This motivates me to make sure that I produce code that is maintainable – bug-free – and easy to understand.*" *(Q3. 2016.04_P18)*

Since student contributions are intended to be deployed in practice, students needed to be tasked with **real, novel requirements**:

"*There's a focus on solving problems that may not have been solved before, there's no deadlines so self-motivation is a must, and the primary goal isn't some grade - it's producing working code and benefiting your team.*" *(Q2. 2014.04_P15)*

These requirements contributed to developing students' practical **problem solving** skills:

"*Further, it was one of my first experiences working with a large, existing code base, where my assignments were not spoon fed to me. I had to figure a lot of things out on my own.*" *(Q1. 2015.12_P13)*

Students also perceived **UCOSP as different from the traditional classroom** in their comments:

"*Another difference I realized was that the notion of not being able to find a correct solution given the current stage of various other projects which my project depended on, was in fact ok. This is rarely the case in my other courses as the instructors plan for us to be able to find a correct solution in all of the assignments we have been given.*" *(Q2. 2014.04_P34)*

The **applied** nature of UCOSP was appreciated by students "*Practical experience— which I never got out of any other course.*" *(Q1. 2016.04_P06)*

## 4.3 Real Mentors

Students participating in UCOSP are primarily supervised by mentors from their projects while they participate in the program. These mentors meet the students during the in-person sprint, hold weekly team meetings online (usually with some form of video chat), and have regular out-of-band contact with the students through the normal communication mechanisms used by the project development community including its issue tracker, version control system, and code review system.

Mentors played a key role in helping **guide students** as they tackled their development tasks, "*Having a dedicated mentor also helped me feel supported and gave guidance on how to approach some more difficult problems.*" *(Q2. 2015.12_P05)* The roles these mentors played varied from helping students **learn new development skills** "*The mentorship from our mentors was extremely helpful in helping me solve problems I was stuck on, as well as developing my skills for debugging.*" *(Q6. 2015.12_P11)*, to assisting the students with general **program understanding** tasks, "*Being guided by mentors as well as reading and understanding lots of different code structures.*"

All mentors had the expectation that each student would contribute code to their project that would improve their projects. As such, there was a **strong emphasis on the quality** of the students' solutions:

"*Having good mentors that did code reviews and gave meaningful advice on changes that could be made to improve the code.*" *(Q6. 2014.12_P22)*

Mentorship through **code reviews** played a prominent role in maintaining quality and providing meaningful feedback:

"*I received plenty of feedback (via code reviews from my mentors) that were tremendously helpful for my projects and overall understanding of the codebase.*" *(Q2. 2015.12_P17)*

This also helped students **reason about non-functional aspects of their code**, for example: "*Well written code was more important than just functional code.*" *(Q2. 2014.12_P12)* Students also realized that they were not just churning out code to satisfy a simple assignment, but that code needed to be able to live long term:

"*I found that a common theme was making sure that solutions were*

*as succinct/reusable/readable as possible and it was great to be able to merge code with confidence." (Q1. 2015.12_P26)*

Students believed that their mentors and the rest of the open source development team could facilitate **valuable knowledge transfer** they could use in the future:

"*Working with industry developers Being able to talk with those developers and other student devs Not being told the entire problem or the solution to a problem Self-motivation and self-research Receiving and performing code reviews Weekly Reports (reminds you to get things done!).*" *(Q6. 2015.12_P15)*

Students appreciated the chance to work as **members of distributed teams**: "*... see first-hand both the benefits and hardships of distributed-development " (Q6. 2015.12_P22)* In some cases, mentorship was guided by the mentors but was received from other students and the community at large:

"*I don't think this opportunity is available with anything else; co-op jobs would be the closest, but I've never heard of a remote-work co-op job.*" *(Q6. 2014.04_P15).*

This also helped students hone their **communication skills** that they viewed as transferable to the kinds of situations they expected to experience in the future:

"*It's useful because a lot of development happens remotely (see the gazillions of open-source projects out there) and some companies now exist as remote-only organizations. Learning to function remotely is part of the more distributed culture we're now seeing with the Internet.*" *(Q6. 2014.04_P15))*

Finally, **tangential benefits of mentorship** such as resume building ("*The open source software aspect of the program is very useful since the work I do is public and is linked to my GitHub account.*" *(Q6. 2015.12_P32)*)), networking ("*Most importantly, UCOSP provided me with connections to incredibly talented and seasoned software developers who were willing to mentor me.*" *(Q6. 2015.12_P17))*, and professionalism also appealed to the students.

## 4.4 Other benefits

In addition to the themes listed above, students also appreciated several other benefits that came from this intensive project course. These fell into two main themes which we term *Soft Skills* and *Technical Skills*.

*4.4.1 Soft Skills.* Working on real tasks for real projects forced the students to collaborate with their mentors, fellow students, and the community increasing their **practical communication skills**:

"*My collaboration skills greatly increased and confidence when working on team projects has improved.*" *(Q1. 2014.12_P38)*

Because they could see their concrete value, these communication skills were valued by the students and gave them specific applied settings in which to use them:

"*I also improved my communication skills as a software developer, as it's necessary to clearly explain what you're trying to do, the roadblocks you're facing, and the strategy you intend to use to overcome these obstacles.*" *(Q1. 2015.12_P17)*

The communications skills that were required in UCOSP were also viewed positively in comparison to students' other courses, "*UCOSP was one of the few courses I have ever taken that promotes teamwork and collaboration— important skills in the real world.*" *(Q2. 2015.12_P02).*

For students who had not had prior co-op experience, being forced to **collaborate with others** using community-enforced mechanisms was viewed as an important skill:

"*It was a great opportunity to meet students from across Canada and to work in a geographically distributed group I gained much experience in working with version control systems, collaborating with others, and found it particularly rewarding to work on an open source project that benefits the community.*" *(Q1. 2014.04_P39)* They learned the value of the tools through their experience "*Since we used pull requests for everything— keeping up with what was going into Waterbear was quite easy.*" *(Q5. 2014.04_P12).* Students also appreciated learning the **collaborative processes** that they used "*namely branching, merging, and issuing pull requests*" *(Q1. 2016.04_P18)* and learned how to use them effectively "*keeping your working copy up to date is crucial. Resolving a conflict of 10 lines of code is about 10000 times easier than resolving one with 100*" *(Q8. 2014.12_P36)*

By engaging with real users, their **motivations for engaging** heavily in the program were also enhanced because they could clearly see the novelty of their work:

"*It's nothing like a traditional sit down class where problems are contrived, requirements are well defined, and the problem is easily malleable to your use case.*" *(Q2. 2015.12_P28)*

Working in a professional team motivated students to produce quality contributions: "*it motivated me to give better documentation and communicate more clearly and efficiently*" *(Q3. 014.0_P01)*

Students noted that UCOSP provides an experience that mimics a job and they recognize the value of this experience:

"*UCOSP motivation is more closely related to the kind of motivation we use at our jobs: peer approval – excitement for new ways of solving the problem – working for the good of entire project \ team.*" *(Q3. 2014.04_P08)*

Finally, given the novel nature of the tasks, their complexity and time requirements were often unclear at the outset of the tasks forcing the students to **carefully manage their time**:

"*It is different in that it can't be put off to the last minute in the same way other courses can: even if you don't run into any problems, you need to work steadily and be in communication with your supervisors and team members, especially if they are relying on you to finish something so they can move forward.*" *(Q2. 2014.12_P21)*

Students also noted that **time estimation** ended up being harder when performing novel tasks:

"*I tend to underestimate the time that tasks take because I view them in the best light with all uncertainty easily avoidable by either Googling or asking questions, and full days to work on them.*" *(Q1. 2014.04_P33)*

By integrating with teams, students seemed to gain greater appreciation of the need for following established communication and development processes. The benefits of practicing these skills in a way they would be able to describe in a job-interview setting (or on a resume) provided a wealth of motivation to the students to fully engage in their work and try to develop future professional contacts.

*4.4.2 Technical Skills.* As one would expect, students learned a range of technical skills from working on their projects. These included **languages** "*a new language (Go)!*" *(Q1. 2014.04_P11)*, **test frameworks** "*Python based unit test frameworks*" *(Q1. 2014.12_P14)*, **design patterns** "*MVC pattern*" *(Q1. 2014.04_P27)*, **system administration skills** "*Working in and configuring virtual machine development environments*" *( (Q1. 2014.12_P27))*, and **development frameworks** "*a variety of different web technologies such as Ruby on Rails*" *(Q1. 2014.12_P38)*,

Students also discovered which languages, tools, and frameworks were being used in practice, and could see the impact that these tools and processes had on the quality of their work.

## 5 DISCUSSION

In this section we will discuss curriculum implications of our results and threats to validity.

### 5.1 Curriculum Implications

The most interesting codes in our analysis fall under the *Real Projects*, *Real Tasks*, and *Real Mentors* themes. Crucially, these three themes have several crosscutting concerns that bind them together. The value that the students get from an experiential software engineering learning opportunity that includes all three aspects of experientialism is much greater than the value they get from an experience that includes only one or two aspects.

*Real processes and tools.* One crosscutting concern is the benefits students accrue through using real development processes and tools. While they are naturally excited about the technical skills they have gained, they are also keenly aware of the skills they have gained in terms of process (e.g., version control strategies, issue management skills) code quality (e.g., code review, guided design discussions), and the importance of communication (e.g., with mentors, other students, users, and other members of the community). While these could be taught in the classroom, experiencing them as a part of a real project reinforces their practical utility for the students. Simply working on real projects does not in itself provide these benefits. By interacting with their mentors and the community the importance of these standards and practices are significantly enhanced for the students.

*Providing real value.* Another crosscutting concern is the independent nature of the work performed by students in UCOSP. Since the work is intended to be novel, and should be pushed into the projects the students are working on, mentors are incentivized to engage the students to ensure the tasks are real, and the quality of the work is high. This also encourages the mentors to ensure the students make a large contribution in order to maximize their productivity. Without students performing these real tasks for them, mentors have significantly less motivation for participating in the program. We also see mentors pre-planning for UCOSP and thinking carefully about the work that UCOSP students can accomplish; several projects have `UCOSP` tags that they place on new features in their issue tracker that they think would be of appropriate size for UCOSP students.

*Impact of mentorship.* The importance of mentorship emerges as central to both of these crosscutting concerns as it binds our three key themes together. Our mentors are embedded in the open source teams that the students join and provide a natural conduit between student members and the rest of the development team. The mentors welcome the students to their communities of practice and, in turn, expect the students to participate using the team's standard processes and tools. At the same time, mentors have a finite amount of time with which to mentor students. This acts as a form of rate limit that bounds the number of students they can welcome onto their projects.

We are not claiming that a subset of these high-level themes cannot be employed independently, although this would entail its own challenges. For instance, students can be exposed to real projects while giving all students the same tasks and traditional instructor- or TA-led mentorship. Engaging real mentors for these synthetic tasks is not likely to be possible without some other type of external motivation to encourage them to participate. It may also be possible to use real projects with students performing real novel tasks, but without explicit project-based mentorship, it is not clear how easily students would actually find joining the community and having their contributions be accepted.

*Creating new experiential capstone courses.* The geographically-distributed nature of the UCOSP teams could not be replicated with only a single institution. However, the three aspects of experientialism could be built into a single-institution course. The contributions of real tasks on real projects could remain the same. While the distributed UCOSP teams created extra communication challenges and motivated the use of distributed tools, this working environment could be achieved if the real mentors and the project's developer community were geographically distanced from the students themselves. We were able to select strong students from each university to participate in UCOSP. At a single institution, the challenge might be having students rise to the required level of independence and then finding mentors willing to repeatedly invest their time into students when some let them down. To mitigate this challenge, an experiential software engineering course could be offered as an elective that requires students to commit to make regular contributions in order to pass the course.

### 5.2 Threats to Validity

Our work relies heavily on codes that we generated ourselves to study a program in which we are invested. This leaves us susceptible to unconscious bias. Another threat to internal validity is the possibility that students did not answer survey questions honestly. In order to guard against this, the survey answers were anonymous to those responsible for student grades and we did not examine the anonymous data until after the relevant semester was over. Even though students were informed of this, some may still have given answers that they felt we wanted to hear. Because we did not measure any direct outcomes (i.e. all our data is student perceptions of their experience or their improving skills), we are particularly susceptible to student bias.

One threat to the external validity of our study is its generalizability to a broad student population. Study participants came from many different sized universities (small regional schools to large research-intensive institutions), however, the UCOSP program only accepts very strong students from each of these schools as identified by their home faculty liaisons. These strong, self-motivated

students can handle real tasks and usually manage to make an actual contribution to the OS project within the span of the course. Some UCOSP alumni continue to contribute to the OS community for their project even after the program is finished. This motivates and excites the mentors and keeps them engaged and willing to donate their time. It is not at all clear that this model would work with a general student population.

This paper is entirely based on students' perceptions of their UCOSP experience. However, students may not have enough information to understand the many trade-offs involved in curricular decisions. The UCOSP program is expensive and it reaches a limited number of students. The top students who are selected to participate and consequently receive a disproportionate share of resources are not in a good position to evaluate whether these resources could be better deployed in another way. Although UCOSP has benefited over 700 students, this is over multiple years. The program does not scale to the current demand for undergraduate CS education.

## 6 CONCLUSIONS

Capstone projects are often proposed to help bridge the gap between traditional classroom experiences and the skills students need to succeed in the workplace. We designed the Undergraduate Capstone Open Source Projects (UCOSP) program to help students from over 30 Universities across Canada gain real-world development experience. UCOSP has been running for 8.5 years and more than 700 students have participated. By surveying 167 students from 18 universities over 6 terms, we tried to discover what students felt they learned from UCOSP. Ultimately, students valued performing *real tasks* on *real projects* under the guidance of *real mentors*. These three dimensions provided students a wealth of experience they had not received from traditional classroom settings. Our goal is for the specific feedback identified in this paper to be useful for future curriculum designers as they create new experiential capstone projects. UCOSP demonstrates that a program that provides all three of experiential dimensions is possible and student feedback shows that such programs are also a valuable compliment to traditional classroom instruction.

## ACKNOWLEDGMENTS

We are grateful for the financial support that UCOSP has received from Google, the Canadian Association for Computer Science, CIPS (Canada's Association of Information Technology Professionals), and the Jonah Group and for the in-kind contributions from Shopify, Mozilla, Facebook and O'Reilly. We thank the students and mentors for their contributions to the program.

## REFERENCES

[1] Eric Allen, Robert Cartwright, and Charles Reis. 2003. Production Programming in the Classroom. *SIGCSE Bull.* 35, 1 (Jan. 2003), 89–93. DOI: http://dx.doi.org/10.1145/792548.611940

[2] Craig Anslow and Frank Maurer. 2015. An Experience Report at Teaching a Group Based Agile Software Development Project Course. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 500–505. DOI: http://dx.doi.org/10.1145/2676723.2677284

[3] María Cecilia Bastarrica, Daniel Perovich, and Maíra Marques Samary. 2017. What Can Students Get from a Software Engineering Capstone Course?. In *Proceedings of the International Conference on Software Engineering: Software Engineering and Education Track*. 137–145. DOI: http://dx.doi.org/10.1109/ICSE-SEET.2017.15

[4] Andrew Begel and Beth Simon. 2008. Struggles of New College Graduates in Their First Software Development Job. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 226–230. DOI: http://dx.doi.org/10.1145/1352135.1352218

[5] William Billingsley and Jim Steel. 2013. A Comparison of Two Iterations of a Software Studio Course Based on Continuous Integration. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*. 213–218. DOI: http://dx.doi.org/10.1145/2462476.2465592

[6] Aaron Bloomfield, Mark Sherriff, and Kara Williams. 2014. A Service Learning Practicum Capstone. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 265–270. DOI: http://dx.doi.org/10.1145/2538862.2538974

[7] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. 2015. Software Engineering Project Courses with Industrial Clients. *Trans. Comput. Educ.* 15, 4, Article 17 (Dec. 2015), 31 pages. DOI: http://dx.doi.org/10.1145/2732155

[8] Heidi J. C. Ellis, Gregory W. Hislop, Stoney Jackson, and Lori Postner. 2015. Team Project Experiences in Humanitarian Free and Open Source Software (HFOSS). *Trans. Comput. Educ.* 15, 4, Article 18 (Dec. 2015), 23 pages. DOI: http://dx.doi.org/10.1145/2684812

[9] Marisa Exter. 2014. Comparing Educational Experiences and On-the-job Needs of Educational Software Designers. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 355–360. DOI: http://dx.doi.org/10.1145/2538862.2538970

[10] Fabian Fagerholm, Alejandro Sanchez Guinea, Jay Borenstein, and Jürgen Münch. 2014. Onboarding in open source projects. *IEEE Software* 31, 6 (2014), 54–61.

[11] Fabian Fagerholm, Alejandro S. Guinea, Jürgen Münch, and Jay Borenstein. 2014. The Role of Mentoring and Project Characteristics for Onboarding in Open Source Software Projects. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*. Article 55, 10 pages. DOI: http://dx.doi.org/10.1145/2652524.2652540

[12] Reid Holmes, Michelle Craig, Karen Reid, and Eleni Stroulia. 2014. Lessons Learned Managing Distributed Software Engineering Courses. In *Companion Proceedings of the International Conference on Software Engineering*. 321–324. DOI: http://dx.doi.org/10.1145/2591062.2591160

[13] Beanbag Inc. 2018. Review Board. (2018). Retrieved February 9, 2018 from https://www.reviewboard.org/.

[14] David A Kolb. 2014. *Experiential learning: Experience as the source of learning and development*. FT press.

[15] Daniel E. Krutz, Samuel A. Malachowsky, and Thomas Reichlmayr. 2014. Using a Real World Project in a Software Testing Course. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 49–54. DOI: http://dx.doi.org/10.1145/2538862.2538955

[16] Robert McCartney, Swapna S. Gokhale, and Thérèse M. Smith. 2012. Evaluating an Early Software Engineering Course with Projects and Tools from Open Source Software. In *Proceedings of the International Computing Education Research conference*. 5–10. DOI: http://dx.doi.org/10.1145/2361276.2361279

[17] Christian Murphy, Swapneel Sheth, and Sydney Morton. 2017. A Two-Course Sequence of Real Projects for Real Customers. In *Proceedings of the Technical Symposium on Computer Science Education*. 417–422. DOI: http://dx.doi.org/10.1145/3017680.3017742

[18] Andres Neyem, Jose I. Benedetto, and Andres F. Chacon. 2014. Improving Software Engineering Education Through an Empirical Approach: Lessons Learned from Capstone Teaching Experiences. In *Proceedings of the Technical Symposium on Computer Science Education*. 391–396. DOI: http://dx.doi.org/10.1145/2538862.2538920

[19] Tom Nurkkala and Stefan Brandle. 2011. Software Studio: Teaching Professional Software Engineering. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 153–158. DOI: http://dx.doi.org/10.1145/1953163.1953209

[20] Alex Radermacher and Gursimran Walia. 2013. Gaps Between Industry Expectations and the Abilities of Graduates. In *Proceeding of the ACM Technical Symposium on Computer Science Education*. 525–530. DOI: http://dx.doi.org/10.1145/2445196.2445351

[21] Richard A. Scorce. 2010. Perspectives Concerning the Utilization of Service Learning Projects for a Computer Science Course. *Journal of Computing Sciences in Colleges* 25, 3 (Jan. 2010), 75–81.

[22] Therese Mary Smith, Robert McCartney, Swapna S. Gokhale, and Lisa C. Kaczmarczyk. 2014. Selecting Open Source Software Projects to Teach Software Engineering. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 397–402. DOI: http://dx.doi.org/10.1145/2538862.2538932

[23] Anselm Strauss and Juliet M. Corbin. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications.

[24] Eleni Stroulia, Ken Bauer, Michelle Craig, Karen Reid, and Greg Wilson. 2011. Teaching Distributed Software Engineering with UCOSP: The Undergraduate Capstone Open-source Project. In *Proceedings of the Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development*. 20–25. DOI: http://dx.doi.org/10.1145/1984665.1984670

[25] Claudia Szabo. 2014. Student Projects Are Not Throwaways: Teaching Practical Software Maintenance in a Software Engineering Course. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 55–60. DOI: http://dx.doi.org/10.1145/2538862.2538965