Making Team Projects with Novices More Effective: An Experience Report

Carlos Paradis
University of Hawaii at Manoa
cvas@hawaii.edu

Anthony Peruma
University of Hawaii at Manoa
peruma@hawaii.edu

Rick Kazman
University at Hawaii at Manoa
kazman@hawaii.edu

Abstract

Computer Science capstone projects have shifted to offer more opportunities for students to engage with stakeholders as a team to simulate a real-life work scenario. Most works focus on the class design modifications and requirements for both students and stakeholders. However, discussions on how said requirements are instantiated in a concrete software development process that can accommodate both students and stakeholders are only briefly presented, and challenges are presented from a class standpoint. This work is an experience of one project in the capstone course in the Information and Computer Sciences Department at the University of Hawaii at Manoa.

1. Introduction

To adequately equip computer science students for success in the industry, it is crucial to offer them the chance to apply their theoretical knowledge and practical skills to real-world projects, thereby gaining firsthand experience and insights into the challenges and requirements of the industry, fostering a deeper understanding of its dynamics (Tenhunen et al. (2023)). To this extent, most computer science departments in higher educational institutes include a mandatory Capstone Course for final-year undergraduate students that span across one or two semesters.

Like other institutes, the Information and Computer Sciences Department at the University of Hawai'i at Mānoa provides students the opportunity to practice their software engineering skills through the capstone project course - ICS 496[1]. Through this course, students are grouped into small teams and work with a project sponsor in building or enhancing a software application. In this paper, we report on the lessons learned in one such project involving Kaiaulu——an open-source tool for mining open-source software repositories.

Project courses in computer science offer a broad range of benefits that contribute to a well-rounded education and practical skill development. They require students to work in teams and hence develop teamwork, communication and collaboration skills. They involve larger-scale problems than what students have typically tackled in their prior courses. And they can give exposure to industrial tooling and practices. They are, however, quite time-consuming for both teachers and students, and can pose pedagogical challenges in terms of evaluation.

We believe that choosing to build upon an existing project with a coherent and well-documented architecture, as opposed to having the students code something from scratch, provides greater exposure to realistic development and maintenance concerns. Such concerns are absent in most traditional semester-long software engineering courses, which typically focus on having the students create well-specified deliverables packaged as assignments, with little ambiguity or uncertainty. Kaiaulu, on the other hand, was a small but growing real-world project.

Kaiaulu's existing architectural decisions constrained the students——offering less freedom and fewer opportunities for creativity. But in return the students gained a coherent introduction to a real project and realistic, highly structured software engineering processes. The students were required to learn the project's processes and norms, as well as its code base and requirements. In doing so they were introduced to the practical issues that they will face when they join the workforce: interacting with a variety of stakeholders, dealing with revision control systems and issue trackers,

[1] https://www.ics.hawaii.edu/academics/capstone-project/

URI: https://hdl.handle.net/10125/107313
978-0-9981331-7-1

engaging in code reviews, dealing with automated testing and continuous integration, and creating concise documentation.

Lessons learned in this work can serve schools, government, and industry alike, as they can be applied to other capstone projects or summer internships—projects with a constrained time-frame. In such projects the sponsors typically face a trade-off between learning curves—it takes a significant portion of the project time for the students to learn enough to be able to contribute—and obtaining meaningful assets from those students.

This paper is organized as follows: In section 2 we presented related literature. Section 3 then introduces the tool Kaiaulu, which the student team worked on as part of their capstone. The subsequent section 4, Case Study, presents the subset of the class requirements that constrained the scope of the capstone and how they were embedded in the team's software development process. The Discussion section 5 presents the learned lessons of the case study. Finally, we offer our Conclusions and Future Work in section 6.

2. Related Work

There has been significant research in the area of project-centric software engineering courses for more than 30 years, dating back to the ground-breaking work of Bruegge et al. (1991).

Laplante et al. (2019) discuss the evolution of their software engineering capstone course over the span of 12 years. The authors report on the lessons learned and recommendations, such as emphasizing the use of agile methodologies. Paasivaara et al. (2019) report on their experience of having students work on industry projects as part of their capstone project. Through sponsor surveys, the authors report that the course is beneficial for both students and sponsors, with sponsors mentioning recruitment and technology research as one of their primary benefits. In their experience report, Schorr (2020) discusses their capstone source and student feedback. The author notes that in this course, students decide on the problem they intend to solve and the technology stack. However, the author does note that such an approach results in more effort for the class lecturer. Through a student survey, the author reports on the positive aspects of the course, such as student creativity and motivation. In another experience report, Verdicchio (2021) provides recommendations for adopting a classroom-based capstone course to another modality due to disruptions caused by natural disasters or health pandemics. The authors highlight that modality changes are less challenging for projects that incorporate agile methodologies. In their experience report, Daun et al. (2016) show that their teaching approach for an industry-oriented capstone course is more successful with graduate students than undergraduate students for reasons such as maturity and background knowledge.

Mertz and Quesenberry (2018) present a model for student engagement with project sponsors. This model has evolved over 20 years and is suitable for use in class projects and internships. The authors report that while the model includes the generation of software engineering artifacts, the primary benefit is students build relationships with their sponsors and improve their problem-solving skills. Likewise, Schneider et al. (2020) also present a hybrid agile organizational model for capstone projects. The authors report that overall, their model is successful but also highlights areas for improvement. Morales-Trujillo et al. (2022) examined 6,854 peer evaluations from 193 students to understand the quality of the evaluations and how students evaluate their colleagues. The authors report that most feedback students provided was positive and that stronger students tend to provide more detailed feedback than weaker students. Spichkova (2022) discusses their experience redesigning a software engineering project course to make the course more engaging and give students the experience to be work-ready. The author presents a series of learnings based on their experience and student feedback.

In the above-mentioned studies, the authors discuss the benefits of such courses: more realistic and engaging projects, the ability to put the students' largely theoretical experience into practice, and exposure to real-world requirements, tools, techniques, and standards. We fully agree with these claims, as we have observed similar benefits in our own project experience. We add to this body of research a discussion of how the creation, documentation, and dissemination of a clear software architecture for the system being extended has constrained and informed the activities of the students. This, we claim, improves their educational experiences and outcomes: the constraints of the architecture are a good thing as they provide guidance, context, and structure to the students' learning experiences.

3. ICS 496 - Capstone Project

Introduced in Fall 2022, ICS 496 is a mandatory one-semester, 3-credit, course for final-year undergraduate students enrolled in the general B.S. Computer Science program. The course enables students to work in a team environment and utilize the knowledge gained throughout their academic journey

in planning, executing, and delivering a software project. The course is designed to nurture critical thinking, problem-solving skills, teamwork, project management proficiency, and adept communication skills, emphasizing hands-on learning rather than lectures. The course provides students with practical exposure to real-world software development, equipping them with the necessary skills and understanding to meet industry demands and expectations.

3.1. Course Structure

Despite the absence of traditional lectures, the course is allocated two 75-minute timeslots per week. The scheduling serves two primary purposes. Firstly, it guarantees that students working on the same project have a shared designated time to meet and collaborate or engage with their sponsor (i.e., client). Secondly, these timeslots facilitate student project presentations, as elaborated in Section 3.4.

Furthermore, in addition to presentations, each team must create a project poster they present at the end of the semester. Student grades are based on the quality of their class presentations and sponsor feedback.

3.2. Project Sponsors

The course projects are sponsored by various entities that include the faculty of the department, faculty from other departments within the university, state/federal agencies/departments (e.g., US Army), non-profit organizations, or industry partners. Before the semester begins, sponsors submit project proposals to the course instructor, who reviews and shares them with the students. As part of the proposal submission, sponsors are required to review and acknowledge an expectations document, which we discuss in Section 3.3. Projects include enhancing existing software systems or building new software systems. These systems can be desktop, mobile, or web applications. Project assignments are determined by the course instructor, taking into account factors like students' preferences, skills, and the sponsor's needs.

Throughout the project lifecycle, students work closely with their sponsors, engaging in regular communication, collaboration, and demonstrations to ensure that the project aligns with the sponsor's expectations and requirements. The project sponsor offers guidance, feedback, and assistance in addressing any project-related queries or challenges that arise. This approach allows students to gain valuable experience in stakeholder collaboration and communication, which are crucial for the success of real-world software development projects and their own professional development. Students are expected to hold at least one weekly meeting with the project sponsor and one internal meeting. During the sponsor meetings, students are encouraged to demonstrate their progress, obtain feedback, and discuss requirements and technical matters. Additionally, students sometimes utilize instant messaging platforms like Discord for ad-hoc communication with the sponsor.

3.3. Student & Sponsor Expectations

Unlike in a traditional course, where the students primarily interact with the course instructor, ICS 496 places significant emphasis on fostering student-sponsor interaction. Therefore, to ensure a mutually beneficial experience for both students and sponsors, individuals from each group must review and acknowledge a defined set of expectations and responsibilities. Below is a summary of these expectations.

3.3.1. Student Expectations Students must represent the department professionally, adhere to software engineering best practices, and maintain design and code quality standards. Students should use version control and follow a standard software development process, hold weekly internal team meetings and sponsor meetings, and respond promptly to sponsor inquiries. Students must ensure that project confidentiality is maintained and obtain sponsor approval for code and artifact sharing. The team is responsible for project planning and tracking, issue tracking, and risk mitigation planning. They should also promptly report sponsor-related issues and participate in feedback meetings with the course instructor.

3.3.2. Sponsor Expectations Project sponsors in this course are expected to adhere to several guidelines. They should ensure that students focus on designing, implementing, and testing software systems rather than performing non-software development tasks, like help-desk duties or reading/writing research papers. Sponsors should provide clear project requirements, actively participate in weekly status meetings, and respond to team inquiries within designated timeframes. Collaboration in risk mitigation and granting access to essential resources (including data and systems) are also expected. Sponsors must acknowledge that students are full-time undergraduates who must balance project commitments with other academic obligations. Finally, sponsors must promptly communicate any team-related issues to the course instructor and complete end-of-semester student evaluations.

3.4. Project Presentations

Throughout the semester, students have designated checkpoints where they deliver presentations showcasing their progress, discuss project risks and issues, and share software development best practices and project management techniques utilized in their projects.

The initial project presentation takes place approximately two weeks into the semester, serving as an opportunity for students to showcase their understanding of the client's business problem, project requirements, and the chosen tools and processes. Mid-semester, students engage in the second presentation, during which they discuss their achievements, ongoing tasks, as well as project risks and issues they have encountered. Finally, at the end of the semester, students deliver their final presentation. This presentation not only highlights their accomplishments but also includes a reflection on the obstacles faced and valuable insights gained throughout the course.

3.5. Course Progress

In this subsection, we reflect on the progress of the course, such as sponsor and student feedback. While this course is relatively new compared to other departmental offerings, we are receiving encouraging feedback from sponsors and students. Our project sponsors have generally been pleased with the course and students on the team (for example, refer Quote 1), with one sponsor maintaining engagement over multiple semesters. However, a common concern that has arisen is the course duration, as most sponsors have expressed a preference for it to be extended to two semesters. It is also interesting to note that some sponsors remarked about the learning experience the student gained by working on real-world projects, as shown in Quote 2. Furthermore, sponsor feedback enables the department to identify areas of improvement in the program curriculum and opens the doors for internship and full-time job opportunities for our students.

> *"Capstone projects strengthen our partnership between the University and the DoD, but notably it allowed us to witness ICS student talent and skill."*

Quote 1: General sponsor feedback.

> *"It was a pleasure to work with <student> for the past 3 months on the <company> Internship program...Although he already has some development experience, I think <student> with this project got a taste of team-based development in an MVC application utilizing Agile team philosophy."*

Quote 2: Sponsor feedback about student improvement.

Through project reflections and course evaluations, students highlighted that their learnings were not limited to technical concepts. Soft skills, especially effective communication, and teamwork, were significant learnings for them, which they had not adequately experienced in prior courses. Additionally, project management and sponsor interfacing were also new experiences for most students, especially the need to adapt and be flexible when sponsors change their requirements (for example, refer Quote 3). Finally, unlike traditional course projects, the capstone course exposes students to the real-world situations and conditions of software development. Furthermore, since sponsor feedback is a major component of a student's grade, students are incentivized to perform at their highest level while cultivating a sense of accountability and professionalism.

> *"Gained the skills and knowledge of how it's like to work for an actual company and how they approach the business/industry side of things vs the technical side."*

Quote 3: Student feedback.

4. Case Study - Kaiaulu

In this section, we provide a detailed examination of the capstone course by means of a case study on a specific sponsor's project. This project the students worked on is called Kaiaulu.[2] Kaiaulu is a tool for automating the mining of software repositories. Researchers and analysts need to mine software repositories, so that they can analyze them, supporting a wide variety of purposes. Mining software repositories often requires the handling of multiple data sources to analyze a project's ecosystem. Minimally, a researcher is required to understand the data source in its native form, acquire it (typically using some API), parse and save it (e.g. as a table of data). Unnecessary overheard is incurred if a tool needs to be purpose-built to accomplish all these steps. A better

[2] https://github.com/sailuh/kaiaulu

way is to automate the acquisition, parsing, and analysis steps by using an existing tool (Paradis and Kazman (2022)). Kaiaulu is an R package that helps with mining data from software development communities and their artifacts (Git logs, mailing lists, files, etc.), to help understand them and to understand how they evolve. Kaiaulu provides:

- Parsers to tabulate and manage the naming of software artifacts (R/parser.R)
- Filters to control the scope of analysis (R/filter.R)
- Downloaders for Issue Trackers, Pull Requests, and Mailing Lists (R/download.R, R/github.R)
- Networks to represent software communities as graphs (R/network.R, R/graph.R)
- Identity Matching to connect artifacts together (R/identity.R)
- Metrics commonly used in software research (R/metrics.R)
- Interfaces to third party tools that can provide further analysis (R/dv8.R)
- Reusable Analysis Notebooks which comprehensively present the above features, and warn about threats and pitfalls based on the research literature (vignettes/)
- A Command-Line Interface for server-side analysis (exec/)
- Project Configuration Files that are readable, shareable, supporting reproducibility (conf/)
- A common, simple data model using tables and a regularized nomenclature which can be combined performing table joins.

As described in Section 3.3, students and sponsors are expected to adhere to a defined set of expectations and responsibilities. Below, we summarize those which impacted the design of the Kaiaulu Case Study:

1. **Task Plan and Task Types.** The sponsor must provide the project team with detailed requirements for the proposed software system. The student team activities should involve the design, implementation, and testing of a software system.
2. **Task Status, Metrics, and Evaluation.** The student team should identify (and document) risks and inform the relevant stakeholders in weekly meetings or as soon as possible. The team should also create and maintain artifacts to show project status, metrics, and other details. Three project presentations to the entire class were required of students, all of which explicitly required evaluation of the tasks performed. In addition, the sponsor must complete an evaluation form at the end of the semester.
3. **Communication and Feedback.** The sponsor must have weekly meetings, respond to questions, review and provide feedback/sign off on artifacts, and mitigate risks associated with the project. The student team should also have internal weekly meetings and provide an agenda to sponsor weekly meetings in advance. One of the students must serve as a point of contact. Both the sponsor and the team should meet monthly with the class instructor and/or report any concerns.
4. **Data Access.** Where appropriate, the sponsor must provide the team with access to data, systems, code, and other artifacts related to the project the team is working on. Students are expected to use or provision a version control system.

In the subsequent sections, we discuss how we realized each of the class expectations.

4.1. Task Plan and Task Types

The task plan was formed with three major milestones using GitHub's milestones interface:

1. **First Milestone**: Because Kaiaulu is an *existing* system, the first milestone was to familiarize students as users. The team was then expected to provide as a deliverable a *user experience report* (described in a GitHub's issue comment), highlighting both specific points of confusion and solutions to them. Subsequently, where examples were found lacking, unit tests were implemented, and a "Cheat Sheet"[3] for the associated Notebook functionality was created.
2. **Second Milestone**: The team was required to implement a Bugzilla[4] Wrapper, and a Crawler Downloader/Parser to make available

[3] https://github.com/sailuh/kaiaulu_cheatsheet/tree/main/cheatsheets

[4] https://github.com/bugzilla/bugzilla

issues and issue comments data from open source projects which use Bugzilla as their issue tracker. The wrapper served to tabulate JSON files downloaded by Perceval[5], while the crawler provided an independent implementation using Bugzilla's API. A Notebook was also required to demonstrate both Wrapper and Crawler functionality. This feature is of value to end users of Kaiaulu because, for example, analyzing a project's bug data enables users to identify the buggiest files in the project.

3. **Third Milestone**: The team was to implement an interface to ArchDia's DV8[6] command line interface, enabling literate programming for DV8. This consisted of a set of wrappers to specific DV8 commands, additional data transformations not available in DV8, and an R Notebook showcasing the data pipeline. The value proposition of this feature is that DV8 enables the architectural analysis of a project's source code and so provided a valuable analysis tool to add to Kaiaulu's arsenal.

We next highlight the rationale behind our choice of milestones which we believe are helpful as guidelines for summer internship situations or other capstone projects:

The **first milestone** provided an excellent opportunity to evaluate if Kaiaulu's existing documentation was sufficient to provide a coherent mental model for first-time users, which is otherwise difficult for system maintainers to assess. It also provided a short time-frame for the first deliverable. In fact, the students' learning curve was the true deliverable here. R language knowledge was mostly required to be able to read R code and write unit tests, alleviating some of the member's learning burden, as not all were initially familiar with R. The team's questions could also be directly addressed by the sponsor.

The **second milestone** provided the first substantive deliverable. The first task—creating a wrapper to Perceval—provided the students an easy entry point, utilizing an existing interface and data files, so the team could understand the existing crawler and data. The second task—creating Kaiulu's own crawler—was more difficult as it required the crawler to directly interacting with Bugzilla's API and address shortcomings identified in the wrapper (which we present in the Discussion section 5). Milestone 2 was also relatively decoupled from the rest of Kaiaulu: Its implementation only needed to conform to the overall architecture, which should have already been clear by Milestone 1. No dependencies to the existing code base were required. However, writing in the R language was now required, and the nomenclature of data fields was expected to conform to other bug tracker downloaders which already existed in Kaiaulu. Again, the team's questions were directly addressed by the sponsor.

Finally, the **third milestone** leveraged pre-existing functionality in Kaiaulu, which challenged the team with respect to their first milestone: If the documentation was improved, navigating the existing code base to find the necessary functions should have been easier. Any documentation that may have been initially missed served as a strong incentive: new functions implemented in this milestone had to be carefully documented. Moreover, unlike the prior milestones, only partial information about the DV8 system was available from the sponsor. The team was required to reach out to the sponsor's collaborators to obtain the necessary information to finish this task. Here, the team had to plan more carefully how to parallelize the acquisition of new information and the coding tasks to avoid a situation where they were unable to make progress while waiting for a response.

4.2. Task Status, Metrics, and Evaluation

To track task status, we set the three milestones around the three presentation deadlines required by the class. Moreover, each milestone was assigned a set of GitHub issues during the semester. For each milestone, one issue served to track the entire milestone's progress, similarly to Jira's Epic issues[7]. The remaining issues tracked specific tasks and were assigned to each team member. GitHub's "Projects" were also used to create a Kanban board to track progress on each issue, facilitating progress monitoring. Since the sponsor tracked task status at the individual level, this allowed the team members to move independently across the three milestones, offering greater flexibility for progress (as milestones were independent of one another), and code revisions. Specifically, team members could opt to move to the next milestone independently of their peers' milestones, or they could choose to assist their colleagues. This gave us the opportunity to evaluate their judgment and rationale on either option depending on the circumstances, which we will further describe in the Discussion section 5.

[5]https://github.com/chaoss/grimoirelab-perceval
[6]https://archdia.com/
[7]https://support.atlassian.com/jira-cloud-administration/docs/what-are-issue-types/

4.3. Communication and Feedback

Beyond the required weekly meetings, which were via video-conference, all development communication occurred on issues and pull requests, by sponsor request. Pull requests were also used to provide a feedback loop between the team's commits and requests for revisions, until the changes were ready for merging. Because the sponsor closely observed the issue progress and code revisions, it was easy to assist students with project risk estimation based on the perceived difficulties of each team member and each task.

4.4. Data Access

Since Kaiaulu is open source and publicly available, providing access to source code was a trivial task. For the third milestone, ArchDia's DV8 provides free educational licenses. Other necessary data could be obtained by executing Kaiaulu against open-source projects. The team's final presentation included, for example, visualizations of their own contributions to the project using Kaiaulu.

5. Discussion

In this section, we discuss lessons learned in each of the project's milestones.

5.1. Milestone 1: User Experience Report, Unit Tests and Cheat Sheet

For this milestone, the team was tasked with Kaiaulu setup on their local machines and the execution of the Social Smell notebook[8]. They were then required to provide a user experience report. To better define the user experience report, the team was provided with the definition of every documentation artifact in Kaiaulu for evaluation with respect to their difficulties when learning the system, which we describe here to contextualize their report:

1. **README.md**: Provided on the GitHub repository, it is the starting point of the documentation.

2. **R Notebooks**: Describe an analysis pipeline using multiple functions from the API. It is often the second documentation file read in R packages for examples.

3. **R Functions Doxygen (API)**: When reading R Notebooks, users may be more interested in creating their own R Notebooks, in which case the individual function documentation should suffice. Every function in the package contains pre-defined documentation fields. Required are a title, short description, definition of every parameter, and optionally the returned value of the function. *Parameter* documentation may hyperlink other functions required to obtain the input. The optional field *See also* documents other functions that use the described function return as input. Finally, *reference* fields can provide additional resources, such as scientific literature for metrics.

4. **Project Configuration Files**: Kaiaulu defines configuration files for project analysis in .yml. They encode assumptions made in the analysis, e.g. the commit hash for the git log, the time window of interest, filters used and more.

5. **Wiki**: The wiki provides an extension to the README.md file, such as guidance to download and install third-party software or further background information on project configuration file parameters if the user is not familiar with the analysis.

6. **CONTRIBUTING.md**: This file specifies on GitHub guidance for other users on how to contribute to the project, open issues, ask questions, etc. Since Kaiaulu often receives contributions from students, clarity of this document is of greater importance.

7. **Self-Generated Website**: Kaiaulu utilizes the R package 'pkgdown', which generates a documentation website[9] consolidating all of the information above (except the Wiki and CONTRIBUTING.md).

The experience report deliverable detailed, on a per documentation artifact basis, the perceived deficiencies. Since the detailed report itself is beyond the scope of this work, we summarize our findings instead. The team experienced difficulties with what third-party software tools were required to be installed, and how to proceed if they owned an unsupported operating system (e.g. Windows), or were new to GitHub. Difficulties were also experienced in executing the Notebook in regards to understanding the point of the analysis (the theoretical background was beyond the scope of the Notebook and included in a separate published paper). Finally, the documentation of some functions was found to be too terse.

[8] https://github.com/sailuh/kaiaulu/blob/master/vignettes/social_smell_showcase.Rmd

[9] http://itm0.shidler.hawaii.edu/kaiaulu

Despite the noted difficulties, the students were still able to execute part of the assigned Social Smells notebook even before the first meeting. Some difficulties were expected, due to their lack of theoretical background on the subject of the analysis in the R notebook. This was to be expected. It had been more extensively discussed in prior published work, and addressed during the weekly meetings. While the issues raised on the lack of documentation were reasonable, it also became clear to the team that time constraints limited the level of documentation detail that could be made available. Indeed, the team had to decide against addressing some of the documentation limitations to move forward to the next milestones deadlines, while the more pressing documentation issues were addressed, including providing new examples and unit tests. The documentation issues which remained, nonetheless were documented on the issue tracker to be addressed in the future by the sponsor or other team. To provide further clarity to the Social Smells Notebook, a cheat sheet was also created.[10]

Lastly, a highlight of this milestone was that a bug in a more recent version of Perceval[11] was identified, reported by the team, acknowledged, and fixed by Perceval contributors.

5.2. Milestone 2: Bugzilla Wrapper and Crawler

In Milestone 1, the user experience report was primarily a single group deliverable. For Milestone 2 the group was given a list of tasks to choose from and individually self-assign via one issue representing the second milestone. Subsequently, individual progress was tracked through the milestone. The individual assignment choices, their individual request for feedback, and their difficulties then became more transparent. This assisted the sponsor in ensuring that the group was on track to fulfill the milestones.

While the Bugzilla Wrapper and Crawler were independent tasks, we observed two members of the team ending up with, on the surface, an issue blocker situation. Specifically, the task division required one member to write a parser for another member's downloader function. The sponsor interceded with the blocker by agreeing on a common interface for both the downloader and parser using sample data obtained from Bugzilla's API and documentation, therefore removing the task dependency.

Milestone 2 also familiarized the team with seeking information outside Kaiaulu, in this case, the Bugzilla API documentation. Once milestone 2's first deliverable started to be committed to Pull Requests, a new layer of interaction with the sponsor was introduced: Code revisions. A final highlight of this milestone was that one team member completed the tasks ahead of the others (pending sponsor revisions), and decided to move to Milestone 3 ahead of the rest of the group. This decision would prove vital for the group to successfully complete all three milestones, as we discuss next, and highlights the importance of keeping the milestones as independent as possible.

5.3. Milestone 3: Iterative DV8

Milestone 3 presented the most challenging task for the team. This was due to both the code dependencies of the task on existing Kaiaulu functionality, and also a new situation with respect to team status.

Progress: each team member was now located in a different milestone in regards to their first submission, as their tasks were not entirely completed. For clarity, we refer to each member as M1, M2, and M3 indicating their respective milestones. Once one team member decided to move to milestone 3, again the sponsor provided a set of tasks, and the team members self-assigned the tasks.

As noted in the previous section, M3's decision to move ahead of the group to Milestone 3 provided vital to the successful completion of this task. Specifically, because sponsor knowledge of DV8 was limited, M3 was now required to solicit this information. While waiting for a response, M3 continued to perform revisions for Milestone 2 and Milestone 1. This task alternation process was made possible by the development process being properly documented and separated on GitHub's Pull Requests.

When M3 obtained sufficient information from the DV8 group, the sponsor interceded on the milestone assignment. To decrease project risk, M1 was re-assigned to Milestone 3, and M2 inherited M1's second milestone tasks (as at the time, M2's familiarity with milestone 2 would allow the member to be more efficient). In return, M2 and M1 negotiated that M1 would inherit more tasks from M2 on Milestone 3, as Milestone 1 tasks were deemed easier.

Code dependencies: the set of tasks provided to the team contained varying levels of difficulty, which was made clear to the team during task selection. The simplest task type was a repetition of Milestone 2, requiring only simple system call wrappers. The second type required defining new functionality building on a few wrapper commands. Finally, the third and hardest task required implementing an exporter

[10] Examples of cheat sheets can be found at https://github.com/rstudio/cheatsheets

[11] https://github.com/chaoss/grimoirelab-perceval

function for interoperability with DV8. The last task difficulty required both understanding Kaiaulu functions to generate a graph representation of the data, and DV8's data schema, for conversion. Because all the commands, together, formed a data pipeline from parsing a Git log to analyzing architectural flaws in DV8, an R Notebook and Cheat Sheet was also required.

Surprisingly, the task that required the most revisions was not the implementation of the feature itself, but rather the API documentation. Because DV8 defined its own file nomenclature, which diverged from Kaiaulu's, careful documentation of the R functions was vital so that both users of Kaiaulu and DV8 could understand the intent of a function from each tool's perspective, and also the files it generated. Additionally, function cross-referencing was added via Doxygen in Kaiaulu.

A final highlight of this milestone was the identification of bugs in existing functionality for milestone 3. This was observed in the results of one of the assigned tasks to the students, as we will discuss in section 5.4.

5.4. Sponsor Interference

As noted in prior sections, the sponsor strategically "interfered" with the development process so the team could focus on development and stay on track to accomplish the deliverables. Because we believe this was instrumental to the success of the tasks, we enumerate here a few situations where we deemed interference necessary. For example, the sponsor interfered during the milestone re-assignment to ensure that no team member fell too far behind , or did not gain sufficient exposure across different development tasks.

When the team identified that their deliverable was affected by a bug, the bug fixing was deferred to the sponsor while the team member worked on another task. This provided the opportunity for a unique dialogue between team members and the sponsor on diagnosing an existing bug (beyond the planned scope of the team work) eventually fixing it, and identifying unit tests for them.

Code revisions were extremely helpful in better understanding each team member's strengths and weaknesses, allowing for the sponsor to provide an easier learning curve for all involved. Indeed, at times the sponsor requested one team member who performed the task correctly, to aid another team member who needed help, or requested the group used the weekly individual meetings to compare revision notes to avoid repeating each other's mistakes. Towards milestone 3, the sponsor also incorporated Kaiaulu's GitHub Actions to automatically check that no function documentation was missing fields, which automated some of the revision processes for the team.

A few tasks were removed from the team to ensure that more critical features were finished. For example, the Milestone 3 notebook was re-assigned to the sponsor, so that the end-to-end pipeline could be tested in a timely manner, and revisions were sent to the team before the final deadline. Failure to do so would have resulted in untested deliverables due to time constraints.

In milestone 1, some of the unit tests could not be implemented by the team as they required the assumption of large files. In this case, the sponsor, who was more familiar with the data needed for the unit tests, provided the team with setUp and tearDown functions to generate mock data for unit tests.

Finally, feedback was provided for both team's final presentation and poster.

5.5. Future Process Improvements

While we were able to adjust and improve the process to ensure more exposure to the team for various parts of software development, and to ensure core deliverables could be finished in time, we were unable to implement other ideas in the process as their realization came toward the end of the project. We enumerate them here for future iterations.

During Milestone 3, one opportunity arose for one team member to perform in-line code review for another in their pull request. In future iterations, it would be interesting to request the team to perform one round of code reviews with one another before sponsor code reviews.

In milestone 1, the code review was presented as an issue comment. A better alternative would be to directly refer to the region of the documentation in the issue to facilitate quick inspection. An even better alternative would be to leverage GitHub's ability to comment inline directly on the file.

We found some of the issues became overly long when consolidating information from third-party tools (one reached over 60 comments!). At this point we needed to adopt a convention of 'reply to <hyperlink post> to keep track of multiple threads of discussion with the same issue. We found GitHub's issue interface lacking in that regard. Interestingly, the relatively new "Discussions" section of GitHub allows for threaded discussion. In future iterations, we may use the Discussions space more heavily for clarity.

6. Conclusions and Future Work

In this work, we discussed the ICS 496 Capstone project course, and presented an experience report of using Kaiaulu to instantiate a specific project. We believe the choice of an existing tool with a well-documented architecture provided unique opportunities to employ a development process that can be both rewarding as a learning experience for students, and also offer useful deliverables to stakeholders. While the use of software development on GitHub using commits, issues, pull requests and Kanban boards is not novel in itself, the contribution of this work was centered on the definition of tasks.

We feel that our focus on having students work on an existing system has real benefits. It is more realistic than the kinds of greenfield, small-scale development that characterize the majority of the assignments that an undergraduate computer science major undertakes. Working on an existing system provides benefits and challenges similar to what a student will experience when they enter the job market. Having said this, we do plan to experiment with greenfield projects in the future, to compare and contrast.

Specifically, the tasks not only provided exposure to various software development processes but also provided progressive learning challenges. This was compatible with a constrained time schedule and realistic considering the experience of the team members. We also discussed specific learning experience challenges beyond just implementation, including soliciting information outside the project and being resourceful and adaptable. An added bonus of this project is that, since Kaiaulu is open source, not only the code but the development process is also available for the students to provide in their resumes.

We believe that these are important characteristics for any project-based course in software engineering: 1) appropriately small to start so that students can learn and do something in a relatively short time period, 2) the work should scale incrementally so that the student's increasing knowledge and confidence are matched by ever-more challenging tasks, and 3) realistic eventually; that is, the final product should have complex scope. These characteristics balance the needs of pedagogy with the needs of industrial relevance.

References

Bruegge, B., Cheng, J., & Shaw, M. (1991). *A software engineering project course with a real client* (tech. rep. CMU/SEI-91-EM-004). Carnegie Mellon University.

Daun, M., Salmon, A., Weyer, T., Pohl, K., & Tenbergen, B. (2016). Project-based learning with examples from industry in university courses. *29th Intl. Conf. on Software Engineering Education and Training*, 184–193.

Laplante, P. A., Defranco, J. F., & Guimaraes, E. (2019). Evolution of a graduate software engineering capstone course—a course review.

Mertz, J., & Quesenberry, J. (2018). A scalable model of community-based experiential learning through courses and international projects. *2018 World Engineering Education Forum*, 1–6.

Morales-Trujillo, M. E., Galster, M., Gilson, F., & Mathews, M. (2022). A three-year study on peer evaluation in a software engineering project course. *IEEE Transactions on Education*, *65*(3), 409–418.

Paasivaara, M., Vanhanen, J., & Lassenius, C. (2019). Collaborating with industrial customers in a capstone project course: The customers' perspective. *41st Intl. Conf. on Software Engineering*, 12–22.

Paradis, C., & Kazman, R. (2022). Building the MSR tool Kaiaulu: Design principles and experiences. In P. Scandurra, M. Galster, R. Mirandola, & D. Weyns (Eds.), *Software architecture* (pp. 107–129). Springer.

Schneider, J.-G., Eklund, P. W., Lee, K., Chen, F., Cain, A., & Abdelrazek, M. (2020). Adopting industry agile practices in large-scale capstone education. *42nd Intl. Conf. on Software Engineering*, 119–129.

Schorr, R. (2020). Experience report on key success factors for promoting students' engagement in software development group projects. *2020 IEEE World Conference on Engineering Education*, 1–5.

Spichkova, M. (2022). Industry-oriented project-based learning of software engineering. *Intl. Conf. on Engineering of Complex Computer Systems*, 51–60.

Tenhunen, S., Männistö, T., Luukkainen, M., & Ihantola, P. (2023). A systematic literature review of capstone courses in software engineering. *Information and Software Technology*, *159*, 107191.

Verdicchio, M. (2021). Hurricanes and pandemics: An experience report on adapting software engineering courses to ensure continuity of instruction. *J. Comput. Sci. Coll.*, *36*(5), 150–159.