A toolset to support a software maintenance process in academic environments

Ryan Hardt
Department of Computer and Information Sciences
University of St. Thomas
St. Paul, MN, USA
ryan.hardt@stthomas.edu

***Abstract*—Software engineering and maintenance processes are designed to provide structure and organization around a set of activities involved in the production or maintenance of a software product. Understanding these processes and learning to follow them are important experiences for students in a software engineering course. But it can be difficult for both students and instructors to recognize when a process isn't being followed. Tools designed to guide a process can help. In these environments, tool support can also help ensure that students are using version control systems appropriately while fostering an environment in which students learn from their peers. The importance of tool support for agile processes has been recognized, but little such support has been designed for academic environments. "Co-Op" is a software maintenance-focused process and supporting toolset designed for use in academic environments. The toolset is implemented as a web application that focuses on change impact analysis, use of version control systems that adheres to the process, and communication amongst part-time developers. A brief overview can be seen at https://bit.ly/2YEbWvO.**

***Index Terms*—software maintenance tools, software maintenance process, software maintenance education, software engineering education**

I. Introduction

Learning to follow a software engineering process is an important outcome for students in a software engineering course [1]. Software engineering processes like Scrum were designed for environments with significant differences from the classroom, so instructors typically modify these processes to fit the environment [2]–[4]. Changes to processes commonly used in industry must account for the desired learning outcomes in the course, the part-time nature of the development teams, and the varying levels of experience and motivations of student developers.

The idea of adhering to a process for software development is new to most students in an introductory software engineering course. As a result, it may not be clear to them (or to instructors) when they are or are not performing activities that violate the process. Software engineering tools designed to support a particular process can help ensure that developers follow that process. In the classroom, for example, a student might identify a bug fix or feature update as "complete" before the corresponding code has been committed to the repository, tested, or reviewed. If the process intent is for some or all of these activities to be performed before a task is "completed", then the task is, in fact, not "complete". A tool that supports the process can help enforce constraints like these.

"Co-Op" is a software maintenance-focused process and supporting toolset designed for use in academic environments. It focuses on software maintenance rather than engineering a new system to allow more time in early process phases for activities that facilitate the learning process. Figure 1 illustrates its process model. The Co-Op toolset is implemented as a web application. It integrates with GitHub and GitLab [5] repositories to enforce repository usage that adheres to the process. It also integrates with Slack [6] to facilitate communication with teammates. Source code and installation instructions are present at https://github.com/ryan-hardt/Co-Op.

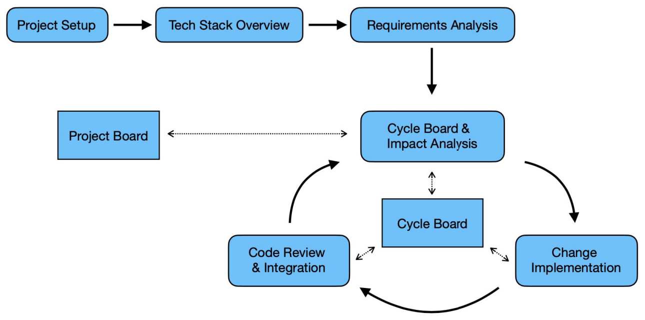


Fig. 1. Co-Op process model.

II. Related Work

Software engineering processes have been designed for use in academic environments, though not focused on software maintenance. Bruegge et al. [7] found that their Scrum-based software engineering process, "Rugby", and their scenario-based design approach, "Tornado", significantly increased students' technical and non-technical skills with respect to software engineering and configuration management. Rugby incorporates structured meeting management to account for the part-time nature of student development. Alfonso et al. [8] found that use of their agile process model effectively drove the learning process and was a valuable and highly satisfactory experience for undergraduate students and their instructors. Their approach made instructor feedback a formal part of the process and included artifacts to facilitate assessment.

Others have advocated for a maintenance focus in software engineering courses. Gallagher et al. [9] illustrated how an introductory software engineering course can be turned into a large software evolution exercise and still provide students with the foundation desired by a traditional software engineering course. Gokhale et al. [10] compared a maintenance-centric version of a software engineering course with a more traditional design-centric version using the same textbook and programming language and showed that students in the maintenance-centric version had a better appreciation for program comprehension and software documentation. Pierce [11] argues that incorporating maintenance exercises in a project-based software engineering class allows students to practice aspects of software engineering that are difficult to experience in a from-scratch project.

III. Implementation

The Co-Op software is implemented as a Java web application using the model-view-controller design pattern. It uses Spring [12] web application frameworks, Apache Maven for dependency and build management, and Hibernate [13] for object-relational database mapping. It is designed to use an embedded Apache Derby [14] database for testing purposes when run locally and a MySQL database for production use. Co-Op integrates with existing git repositories and Slack workspaces. Its architecture is illustrated in Figure 2. A high-level class diagram of its data model can be seen in Figure 3.

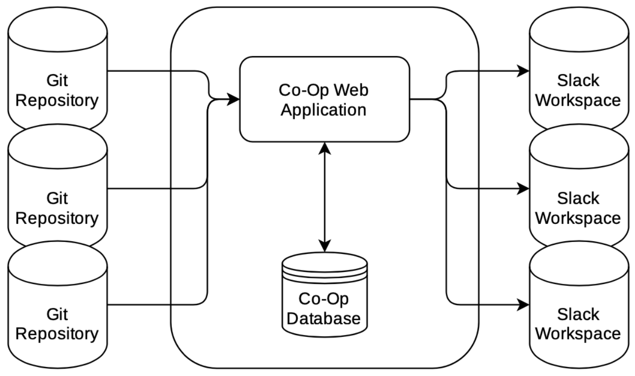


Fig. 2. Co-Op system architecture.

IV. Co-Op Software Usage

A. Project Setup

Co-Op users must first create an account. A user's home page allows one to access associated projects or create a new project. Additionally, this home page provides direct access to one's tasks as well as reports related to one's contributions across all assigned projects. These reports are described in Section IV-C.

When a project is created, it must have a remote git repository assigned. A remote repository host can be associated with a user by indicating the repository type (GitLab or GitHub), its URL, access token (which is encrypted in the Co-Op database), and namespace. After a GitHub or GitLab repository host has been associated with the project, Co-Op retrieves a list of git repositories accessible by the user at that host, one of which must be associated with the Co-Op project. From the project creation screen, a project owner can assign Co-Op users to the project and optionally designate them as "Owners", allowing them to make project-level changes.

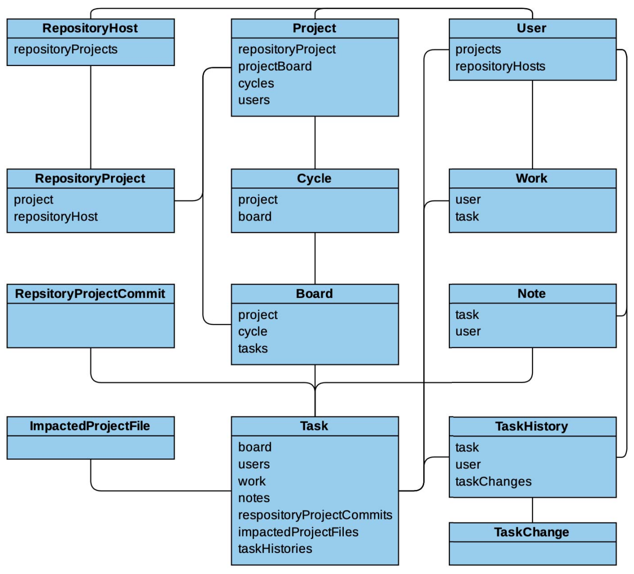


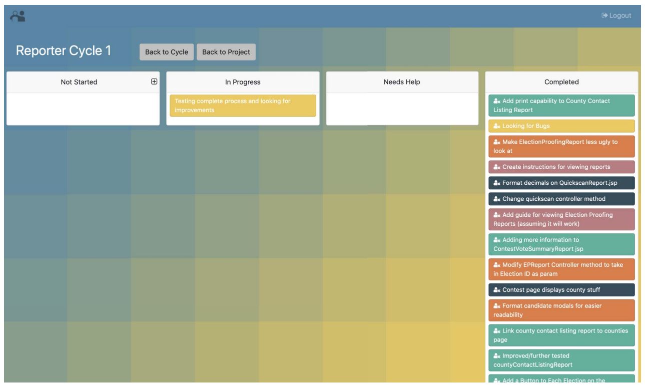
Fig. 3. Co-Op data model class diagram.

From the project screen, a user can access the "project board", which is similar to a "product backlog" in Scrum. This board contains the goals for the project specified as high-level tasks. The project page also allows members to view the remote git repository, view the associated Slack workspace, access project "cycles" (which are similar to "sprints" in Scrum), view the users assigned to the project, and generate project-level reports (described in Section IV-C).

In Co-Op, a "cycle" is a short, fixed-length timeframe in which a team implements a set of tasks. Unlike a sprint, a cycle is not meant to produce a "potentially shippable product", does not have "daily standup meetings", and emphasizes impact analysis, code review, and integration. These differences allow cycles to cope with the varying skillsets, motivations, and availabilities of student developers and to account for the maintenance focus of the Co-Op process. Project owners can create cycles and set their start and end dates. A cycle page displays this information, along with cycle-level reports and timeline, all of which are described in Section IV-C. Importantly, this page allows users to access the "cycle board", which is similar to a "sprint backlog" in Scrum. An example of a cycle board can be seen in Figure 4 and is described in more detail in the next section.

B. Cycle Board

A cycle board consists of four columns of tasks, each of which correspond to a task status: "Not Started", "In



Testing complete process and looking for improvements

Add print capability to County Contact Listing Report

Looking for Bugs

Make ElectionProofingReport less ugly to look at

Create instructions for viewing reports

Format decimals on QuickscanReport.jsp

Change quickscan controller method

Add guide for viewing Election Proofing Reports (assuming it will work)

Adding more information to ContestVoteSummaryReport jsp

Modify EPReport Controller method to take in Election ID as param

Contest page displays county stuff

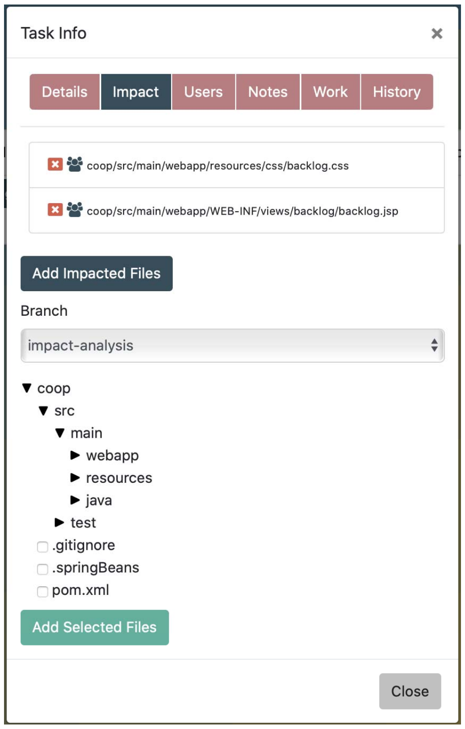
Format candidate modals for easier readability

Link county contact listing report to counties page

Improved/further tested countyContactListingReport

Add a Button to Each Election on the

Fig. 4. Co-Op cycle board.



coop/src/main/webapp/resources/css/backlog.css

coop/src/main/webapp/WEB-INF/views/backlog/backlog.jsp

Branch: impact-analysis

```
▼ coop
  ▼ src
    ▼ main
      ▶ webapp
      ▶ resources
      ▶ java
    ▶ test
  .gitignore
  .springBeans
  pom.xml
```

Fig. 5. Co-Op task impact analysis.

Progress", "Needs Help", and "Complete". A tasks's color indicates its category: "Feature Implementation", "Bug Fix", "Unit Test", "Refactor", "Research", and "Other". Tasks can be added to a cycle board by moving them from the product board or by clicking the "+" button in the "Not Started" task column. A tasks's status can be changed by dragging the task from one column to another, which creates a JSON POST request that updates the task status without refreshing the page in the browser. Similarly, task detail can be updated asynchronously through the task detail view, as seen in Figure 5, which is presented modally over the cycle board when selecting a task. This view allows one to assign a user to a task as an "Owner", "Helper", or "Reviewer". A task must have at least one "Owner". The "Needs Help" column and "Helper" user role make it easy for students to request help with a task and to be recognized for helping others with a task. The "Reviewer" role clearly indicates who is responsible for reviewing a task. Cycle board tasks include icons that identify those to which the viewer has been assigned.

The Co-Op process defines various criteria that must be met before a task's status can be changed, all of which are enforced by the web application. Before a task can be "In Progress", it must have one or more owners identified, a category, time estimate, and estimated completion date. If a task is a "programming task" (one whose category is "Feature Implementation", "Bug Fix", "Unit Test", or "Refactor"), then it must also have a git branch specified before it is "In Progress". The Co-Op software retrieves the names of all branches present in the remote git repository allowing one to be associated with the task.

Co-Op's emphasis on impact analysis requires users to identify all files in the remote git repository that will need to be modified to complete an associated programming task. This impact analysis activity is an important component of the first phase in a Co-Op cycle. Because less-experienced students may conceptually understand *what* needs to be done but not *where* or *how* to do it, this activity, performed as a team, helps clarify the programming responsibilities of the individuals within the team. This functionality is present in the task detail view, as seen in Figure 5. Co-Op retrieves a list of the names of all files and directories in the associated remote git repository and presents them in a collapsible tree of checkboxes. If impacted files have already been identified for the task, they are present above the file tree along with an icon that identifies all users who have previously modified or are currently modifying the associated file. This functionality allows users to easily identify others with whom they may consult regarding file changes.

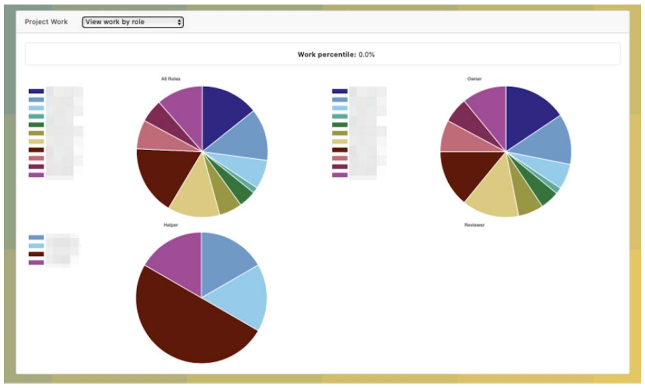
Co-Op's definition of a "Complete" programming task includes an identification of the git repository commits in which the task changes are present. Because programming tasks must have a git branch identified, Co-Op can retrieve a list of commits associated with that branch in the remote git repository. When a user attempts to move a programming task to the "Complete" column, Co-Op retrieves a list of the modified files in the specified git repository commits and compares them to those identified in Co-Op's impacted files list for that task. If the git commits contain file updates or deletions that weren't reported in the impact view, the system will prevent the task from being moved to the "Complete" column and alert the user to the discrepency. If the discrepancies were necessary for the task completion, the user can update the impacted files list in Co-Op, but this activity will be recorded and identified in the "Task History" view. If, however, a commit contained unintentional changes to one or more files, the user can revert that commit and create a new commit containing only the appropriate files.

Additional Co-Op constraints require work to be reported by a task owner before it is “Complete”, require a programming task to be reviewed by another user before it is “Complete” (enforced by checking for work reported by another user assigned to the task as a “Reviewer”), and prevent task modifications outside of the cycle start and end dates.

Whenever changes are made to a task, they are reported in the “Task History” view, which resides within the task detail view. This information is useful to instructors to see how and when a task evolved throughout a cycle, particularly for grading purposes. Whenever a task’s status changes, the change is reported to an optional Slack channel associated with the Co-Op project. Slack is a work-centered communication application. Slack usage allows students to more easily stay up-to-date with the status of their team’s work, while giving them control over how they are notified. Students can elect to receive notifications regarding all posts to their Slack channel, or they can disable these notifications and check manually. Because students can be viewed as part-time developers on a project, tools like Slack that support asynchronous communication are important for keeping everyone on the same page.

C. Reports

Co-Op aggregates data for each user, project, and cycle, and generates reports that include 1) how much time each user reported working on tasks as owners, helpers, and reviewers, 2) how much time each user reported working on specific task categories like feature implementation, bug fix, etc., and 3) how many tasks each user was assigned to as an owner, helper, and reviewer. An example of this information displayed at the project-level can be seen in Figure 6. This data is visible to students and instructors, allowing students to recognize the contributions made by their peers and for instructors to easily view data helpful for grading purposes.



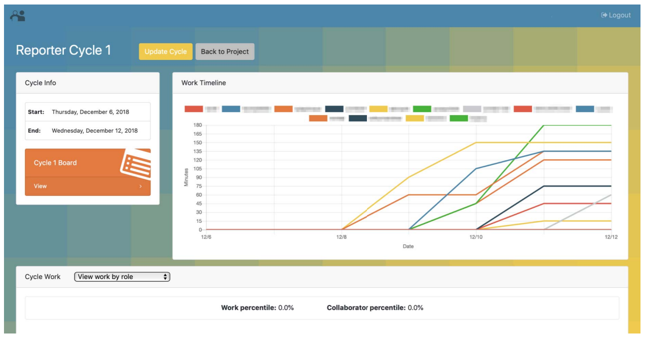
Work percentile: 0.0%

Fig. 6. Co-Op project report.

Cycle reports also include a “cycle timeline” that illustrates when each member reported work on his/her tasks throughout the cycle and how much work was reported. A cycle timeline can be seen in Figure 7.

V. Conclusion and Future Work

The unique characteristics of student-centered software engineering in the classroom call for a process designed for that environment. A project and process centered around software maintenance affords time to focus on software engineering education in addition to product development. A toolset that supports this process can help ensure the process is followed while encouraging student interaction, learning, and growth. A formal evaluation of the Co-Op process benefits and the software’s ability to guide it is being planned. Co-Op will continue to evolve to meet the challenges presented by the changing landscape of software engineering classrooms and work environments.



Start: Thursday, December 6, 2018

End: Wednesday, December 12, 2018

Work percentile: 0.0% **Collaborator percentile:** 0.0%

Fig. 7. Co-Op cycle report.

References

[1] A. Ju and A. Fox, “Teamscope: Measuring software engineering processes with teamwork telemetry,” in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2018. New York, NY, USA: ACM, 2018, pp. 123–128.

[2] C. Matthies, T. Kowark, K. Richly, M. Uflacker, and H. Plattner, “How surveys, tutors, and software help to assess scrum adoption in a classroom software engineering project,” in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE ’16. New York, NY, USA: ACM, 2016, pp. 313–322.

[3] H. Igaki, N. Fukuyasu, S. Saiki, S. Matsumoto, and S. Kusumoto, “Quantitative assessment with using ticket driven development for teaching scrum framework,” in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 372–381.

[4] V. Mahnic, “Teaching scrum through team-project work: Students’ perceptions and teacher’s observations,” *International Journal of Engineering Education*, vol. 26, pp. 96–110, 01 2010.

[5] “The first single application for the entire devops lifecycle.” [Online]. Available: https://gitlab.com/

[6] Slack, “Where work happens.” [Online]. Available: https://slack.com/

[7] B. Bruegge, S. Krusche, and L. Alperowitz, “Software engineering project courses with industrial clients,” *Trans. Comput. Educ.*, vol. 15, no. 4, pp. 17:1–17:31, Dec. 2015.

[8] M. I. Alfonso and A. Botia, “An iterative and agile process model for teaching software engineering,” in *18th Conference on Software Engineering Education Training (CSEET’05)*, April 2005, pp. 9–16.

[9] K. Gallagher, M. Fioravanti, and S. Kozaitis, “Teaching software maintenance,” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 353–362.

[10] S. Gokhale, R. McCartney, and T. Smith, “Teaching software engineering from a maintenance-centric view,” *J. Comput. Sci. Coll.*, vol. 28, no. 6, p. 42–49, Jun. 2013.

[11] K. R. Pierce, “The benefits of maintenance exercises in project-based courses in software engineering,” in *Proceedings Conference on Software Maintenance 1992*, 1992, pp. 324–325.

[12] “Spring makes java simple.” [Online]. Available: https://spring.io/

[13] “Hibernate. everything data.” [Online]. Available: https://hibernate.org/

[14] “Apache derby.” [Online]. Available: https://db.apache.org/derby/