SOFTENG 310: SOFTWARE EVOLUTION AND MAINTENANCE

ASSIGNMENT 1

University of Auckland
Semester 2, 2025

1 ASSIGNMENT DEADLINES

1 Aug 6:00PM: Project Proposal due (Team)
22 Aug 6:00PM: Project due (Team)
25 Aug 1:00PM: Demos (Team, in person)
19 Sep 6:00PM: Peer review due (Individual)

2 LEARNING OUTCOMES

By completing this assignment, you should demonstrate your ability to:

- create a well-documented project,
- follow the required GitHub workflow,
- write code that is easy to understand and maintain,
- continuously evaluate code quality,
- manage dependencies, and
- coordinate your development activities.

3 ASSIGNMENT DETAILS

Groups: This is a group project. Each student must complete this project in a team of ~6 students. Students should self-organise into teams by joining the existing groups on the People page in Canvas. Groups must be formed by the end of week 1. Students who have not joined groups by this time will be automatically assigned. Each team should choose a group name (keep them classroom appropriate please).

Overview: This assignment involves creating and contributing to an Open Source Software project hosted on GitHub. The project must use Python, Java, and/or JavaScript.

Starting the project: Each group must come up with a new software product to develop. Ensure the scale of the project is suitable enough to accommodate approximately 5-6 contributors over the full semester (another group will continue the project in A2). If you are unsure if your product idea is appropriate, discuss with the teaching staff. After identifying the software product that will be developed, you need to create a project proposal document.

Project proposal: The proposal should have the following:

- brief project description including your team's name and members' names and what the proposed software product intends to accomplish.
- list the key software functionalities (high-level features) including in which release (A1 / A2) each feature is intended. The entire team should agree on this set of features.
- selected technology stack.
- completed team agreement (see ECSE Team Agreements.pdf).

- link to a GitHub repository created for A1. The repository should be public. One GitHub repository should be created per group (consider creating a group GitHub account to host the repository that all group members can fork – see below on contribution workflow).

Project documentation: The repository should have the following documentation:

- brief project description (include your team name and a note that this project is associated with the University of Auckland SOFTENG 310)
- license
- Code of Conduct
- README
- Contributor guidelines
- bug report and feature request templates
- list of tasks, including work envisioned for A2 (documented as issues)
- Wiki

See the Documentation module for how to set up your documentation and what to include in your license, code of conduct, README, Contributor guidelines, templates, and issue reports.

To create the list of tasks, a new issue should be created for each envisioned feature described in the project proposal. You can also use issues to track documentation tasks (like setting up the wiki). Ensure to use labels to categorise your issues. If you are using customized labels beyond the defaults provided by GitHub, describe these in the Contributor guidelines.

The wiki should provide any additional project documentation needed including a list of contributors (students in the group with their associated GitHub usernames), a list of contributions made by each student, and minutes from any group meetings.

Contributing to the project: After the repository has been created and the basic documentation established, the team should start working on getting a first iteration of the project up and running.

Types of contributions: You can contribute to your assigned project in various ways:

- ***Contribute code and documentation.*** The primary contribution will be developing and documenting the product. All code or documentation contributions should be associated with an open issue. If an issue does not exist, you should create one first (see guidance below). All additions or modifications to code should include associated tests. All code contributions must be peer reviewed, so ensure enough time remains before the deadline to allow for someone else in the team to perform this review.
- ***Code reviews.*** All pull requests must be reviewed by at least one other team member. To prevent breaking builds, the code review should include running the test suite, running the code, and ensuring the code works as expected. If bugs or other issues are found during the code review, these should be fixed before the pull request is accepted (a new issue does not need to be created). After review and any required changes are incorporated (and approved by the reviewer), someone on the team can merge the pull request. Do not merge pull requests until appropriate approval has been obtained (this will result in a loss of marks for not following the prescribed workflow – see important note below about making mistakes). Prior to merging, all commits should be squashed and merge conflicts should be fixed.
- ***Create a new issue.*** You can create a new issue to document bugs (including documentation issues) or request new features that were not included in the initial list of issues. If you think you have found a bug, first make sure that you are testing

against the latest version. Use the templates for creating new issues. For all new issues, please make sure you check the open issues first to avoid creating duplicate issues. Note: it is *not* required that every issue is completed as part of this project, so some open issues may remain and there should be a suitable number of issues left for A2.

- ***Approving Issues.*** Newly created issues (bug reports or feature requests) must be approved by the team (before anyone begins to work on it). New issues should be checked to ensure bug reports are reproducible, feature requests are appropriate for the product, no duplicate issues already exist, and dependencies to other issues are flagged appropriately. If any information is missing from a new issue, the submitter should be asked for more details. To approve of issues, the team can discuss new issues at group meetings, delegate issue approval to a subset of the team members, or require a certain number of team members to comment with their approval. The approval process should be documented in the project documentation.

Multiple contributors: If more than one person works on an issue, ensure this is documented in the issue and pull request comments. All contributors should be listed on the wiki page describing the contributions.

Workflow: Each group should follow this workflow:

- Fork and clone the main project repository
- Create a new feature branch every time you start work on a new issue. Feature branches should have descriptive names, using a consistent naming scheme. You should branch from main.
- Make your changes in the feature branch in your local clone. If the change is large, consider using a feature flag and multiple pull requests (see Feature Branching and Feature Flags module on Canvas).
- Update your local repository with the most recent code from the main repository. Remember to rebase often – don't wait until all changes have been made.
- Test your changes – run all existing tests and any new tests you have created, run the code, and make sure the application works as expected.
- Create a pull request. The pull request should have a title which summarizes the changes. The body should provide more details about what changes have been made and should reference the number of the associated issue, e.g., "Fixed synchronization issue closes #123". The title of the pull request should not just reference the issue number – it should succinctly describe the actual changes.
- Another team member must review and approve of the pull request.
- Once the reviewer approves the pull request, you (or a delegated team member) can merge your contribution into the main repository. Your team may give all contributors merge access or decide to allow only a subset of team members to merge contributions. This should be agreed upon by all team members and documented in the project documentation.

Code Analysis: You must perform continuous code analysis with Sonar. Make sure to integrate two lines of defense of Sonar into your development workflow to improve code quality, i.e., while writing the code (In-IDE analysis with SonarLint) and after merging your contribution into the main repository (Main branch analysis with SonarCloud). You can also include the second line of defense: before merging any pull request into the main branch (Pull request analysis with SonarCloud), but this is not easy to set up for external pull requests, so it is not required.

Vulnerability Analysis: You must use Snyk to detect and automatically submit pull requests for new vulnerabilities as they are detected, also submit pull requests for known vulnerabilities (backlog). Do not select Fix all vulnerabilities for the same dependency in a

single PR, leave the default behavior of one PR per vulnerability. Merge pull requests when practical (breaking changes may prevent some patches from being merged).

Coordination: Make sure you coordinate before you start an issue to avoid multiple students working on the same issue. To coordinate, assign yourself to the issue or comment on the issue and have one of the maintainers assign you. If another group member has already claimed an issue, you cannot work on this issue (unless the issue is large and requires more than one student). Each student can have a maximum of one claimed open issue at a time to avoid cases where students are reserving issues and not actively working on them.

If you open a new issue and it is picked up by another student, you should respond if requests for additional information are made.

All coordination should occur via issue or pull request comments. In addition, you should meet as a group each week. As a result of these meetings, someone in the team should document any discussed changes to the management of the project in the project's documentation. Any discussions related to specific issues, pull requests, or commits should be documented as a comment on the issue/pull request/commit. In all cases, be clear that the comment is recording a discussion that occurred in the group meeting.

Dependencies: It is very likely that there will be technical dependencies between some of the issues on each project. Before creating a new issue or working on a new issue, you should look at the other open issues to see if there are any dependencies that you will need to manage. If you identify dependencies, you should note the dependency in a comment (e.g., "Depends on #123" or "Blocks #123"). If dependencies are found, you should coordinate your changes. Be sure this coordination is documented in the comments.

External dependencies: Use a package manager (e.g. maven, pip, npm) and declare all external dependencies in the correct file (e.g. pom.xml, requirements.txt, package.json).

Amount of contribution: The overall goal of this project is to enable learning. There is no set amount of contribution required. However, to enable learning and to ensure an equitable workload across the team, all group members should make at least two code contributions and perform at least two code reviews.

Escalating Issues: If you see cases of discussions that are not fostering an inclusive environment or any other issues that the team cannot easily resolve, you should bring these to the attention of the teaching team as soon as possible.

Making mistakes: There is a lot of information in this document. You should read it thoroughly before beginning the project. You are expected to adhere to the required workflow. However, the goal of the project is to learn. If mistakes are made early in the project while learning the workflow, please acknowledge these in the wiki so the marking team can take this into consideration (you will not lose points for a one-off mistake). Explain why the mistake was made and what actions were taken to prevent the mistake from occurring again.

Submission: All contributions for this assignment will be completed on GitHub. All contributions must be made prior to the due date. Prior to submission, create a release in GitHub, using an appropriate version number based on semantic versioning principles. Ensure the release contains release notes explaining what is contained in the release (see the Software Reuse and Dependencies module for instructions on releases and semantic versioning).

A link to the project repo should be submitted on Canvas (one submission per group). Also submit any additional files that are required or any other information that is needed to run the project that is not available in the GitHub repo (e.g., .env files, API keys, etc.).

4 DEMOS

Each group will have 10 minutes. Introduce team, explain the project goal, describe tech stack, demo the system and the main functionality. Describe vision for next iteration (that another group will implement in A2). All group members should be present for the demos.

5 PEER EVALUATION OF TEAM MEMBERS

Each team member should actively participate in the group project. You will receive a peer evaluation form from TeamMates. Individual marks may vary based on contribution level as perceived by your peers. Individuals who do not submit the peer evaluation form will lose marks. Please note that some questions on the peer evaluations will be made available to your team members, so the feedback should be constructively written. The TeamMates form will clearly indicate which questions will be shared (anonymously) with the team member and which are private to the teaching team.

6 PEER REVIEW OF ANOTHER TEAM

Each student will be required to peer review another team's repository. You will receive a peer review form from TeamMates. The peer review will consider the extent to which:

- How to contribute and use the project was explained in the project documentation
- Issues were well documented and included all required information
- Issues were well organized with appropriate labels
- Code was well commented, easy to understand, and maintainable
- How easy it is for newcomers to start contributing to the project
- Scope of the project (and remaining future work) is sufficient for another group to continue working on the project in A2

Please note that some questions on the peer reviews will be made available to the other team, so the feedback should be constructively written. The TeamMates form will clearly indicate which questions will be shared (anonymously) with the other team and which are private to the teaching team.

7 MARKING CRITERIA

- Assignment 1 Project Proposal (2%, group)
  - Project goal and high-level features are identified and presented well
  - Project platform and technologies are listed
  - Scope of project appropriate
- Assignment 1 (18%, group)
  - Project includes all required documentation
  - How to contribute and use the project is well documented
  - Issues are well documented and include all required information
  - Issues are well organized with appropriate labels
  - Code is well commented and easy to understand, pull request titles are meaningful, and pull request descriptions provide sufficient detail
  - Code reviews demonstrate significant effort and are constructive
  - Pull requests demonstrate use of the specified workflow

    - Issues are well coordinated; dependencies are managed as required
    - Project scope sufficient for full semester
    - Source code quality is analysed and improved using Sonar
    - Vulnerabilities automatically detected and patched used Snyk (if none detected, provide screenshot of Snyk project showing no vulnerabilities in wiki)
    - Test suites adequate
    - Product quality
    - Product demo presentation
    - Peer evaluations from your team members
    - Constructive peer evaluation provided to your team members
- Assignment 1 Peer Team Review (5%, individual)
    - Constructive peer review provided to another group that demonstrates a thorough review

8 LATE SUBMISSIONS

Late submissions will incur the following penalties:

- 15% penalty for 1 to 24 hours late,
- 30% penalty for 24 to 48 hours late, and
- 100% penalty for over 48 hours late (Canvas assignment automatically closes).

If you have a legitimate reason for submitting late, discuss this with the lecturer well in advance of the assignment due date.

9 USE OF GENERATIVE AI TOOLS

The use of generative AI tools in coursework is permitted only with acknowledgement. Students are allowed to use generative artificial intelligence text and art generation software, such as ChatGPT and DALL.E 2 in this assignment. However, you must reference any use of such tools.

Referencing: For guidance on how to reference AI generated content in your writing, visit Quick©ite.

Please be aware of the limitations of ChatGPT, including the following:

- In order to achieve high-quality results, you must provide suitable prompts. Keep refining your prompts until you get quality outputs.
- Do not rely on any information given by the tool. Unless you can confirm the answer with another source, assume that any facts or figures provided by the tool are incorrect. It is your responsibility to ensure that the tool does not make any errors or omissions, and it works best for topics that you are familiar with.

10 FREQUENTLY ASKED QUESTIONS

Question: When working with feature branches on a fork, would you expect to mirror the feature branch on the upstream, or pull request directly on the main branch?
Answer: You can submit your pull request directly to main on upstream

Question: How detailed of information should we provide on the wiki list of contributions?
Answer: The list of contributions on the wiki does not need to be very detailed. You could, for example, list the features (with a list of associated issues) you implemented and which team members worked on them. You can also list non-technical contributions if needed.

Question: Why do we need to put additional documentation on the wiki?
Answer: If the README, contributor guidelines, and code comments are enough documentation, then you might not need anything additional documentation in the wiki. You will be marked holistically on how well documented the project is (judged by the teaching team and informed by the peer reviews from the group that takes over the project), so if you don’t need any additional documentation on the wiki because the rest of the documentation is sufficient, you won’t lose marks for not having additional documentation on the wiki.