SWEN 6301: Open-Source Excursion

Revision 1

Part A due on October 31st, 2:00pm
Part B due on December 12th, 2:00pm
Part C due on December 12th, 2:00pm

240 Points (30% Overall)

Your high-level goal is to produce and submit a non-trivial modification or extension to an open-source project in a way that maximizes the chances that the project maintainers accept it. Your team will select an open-source project, contribute to the project, and present your insights to the class. You will individually reflect on your teamwork and open-source experience.

Learning Goals

- Holistically apply software engineering methods in the context of a real-world problem, including process, requirements, architecture, implementation, measurement, and quality assurance.
- Gain broad and deep exposure to the culture and practices of open-source communities
- Design an appropriate infrastructure for an own open-source project.
- Engage with an open-source community.
- Identify process issues and suggest improvements in real-world projects, including communication, collaboration, tooling, quality assurance, formal and informal rules and policies.
- Coordinate within a team and adopt practices for efficient teams.
- Understand a project's architecture and design and make a decision about the feasibility of a proposed task.
- Divide and schedule work within a project.
- Discuss how agile practices affect development.
- Discuss business concerns and business models of software development.

Overview

Selecting the Development Tasks

As a team, you will select an open-source project and complete one or more bug fixes or extensions within it. For the rest of this project document, we will refer to bug fixes and extensions as tasks. You have considerable freedom in which project and tasks you choose (see Appendix A for some helpful pointers), so long as they adhere to the following criteria:

- The open-source project should be active, with many contributors. *Do not choose dead or maintenance projects without sufficient community support.* Do not make this mistake. Notice, that also the opposite problem can exist, though it is rare: in a project that is new and very active with many professional developers working on it, it may be difficult to contribute a task before it is completed by others.
- The task(s) should be taken from a bug report or feature request in a public database or message board, following whatever protocol the project uses to communicate and track open issues. Do not invent a task. Address an actual, documented project need.
- The task must require changes to the project's source code. *Pure documentation or design tasks are not appropriate.*

- You may choose one large task or several smaller, related tasks. Choose tasks that benefit from teamwork and are appropriate for your team size (i.e., do not select one small independent task per team member). The tasks should be scoped such that each team member spends around 36 hours each on the project: ~ 8 hours to identify a project and get the lay of the land; ~ 18 hours to create a work list, design, and execute your changes; and ~ 6 hours to reflect and prepare your report and presentation.

Planning the Development Tasks

Plan before you start coding. You should identify risks and requirements and develop a collaboration plan and schedule.

Performing the Development Tasks

As a team, implement the proposed changes/tasks. You should write code and perform adequate quality assurance activities. Beyond that, you may also need to:

- Take further steps to understand the project's code. You might find it useful to engage in intra-team discussions using static or dynamic diagrams. You might also find it useful to elicit feedback on your ideas by communicating with members of the open-source community.
- Submit your changes to the project. Create any necessary documentation to enable acceptance of your code. New contributors rarely have commit privileges to a master repository. Common contribution mechanisms include pull requests, emails to a project lead, or discussion board posts. You may also need to update the bug database.
- You should reflect on the guidelines, concepts and ideas taken in the class to design and implement high-quality contribution.
- Plan and perform an appropriate level of "marketing" for your submission. Avoid stepping on toes and keep your activities appropriate for the project culture. Try to mimic contributors who have previously successfully submitted similar work.
- Solicit feedback and respond to those who take the time to evaluate your work.

You are required to submit your work to the open-source project. It is not required that the project accepts your submission.

Reporting and Reflection

You will report on your project and task selection, work, and experience in several ways (see below). This will include a group presentation to the class.

Deadlines and Deliverables

This project has four (4) deliverables at three (3) separate deadlines.

Part A: Task Selection and Planning (75 points, team due October 31st, 2:00pm)

The first deliverable is an **initial report** on the project and task(s) you select, with a proposed schedule (with estimates).

Start by researching candidate open source projects with an eye towards making an informed decision about which project you will contribute to. As examples, consider: the type of software, the project age, the number of active contributors, the amount of activity and communication among contributors, the number and types of feature requests/bug reports you might address, the tools and mechanisms the project uses to communicate and collaborate, the dominant programming language/paradigm/framework, as well as the larger context in which the software operates. Communication with the candidate open source projects is encouraged.

Your goal is to to make a principled, informed decision as to which project and task(s) you will tackle. The type of information you collect can vary depending on how your team makes this decision. However, you should justify that decision by grounding it in facts about the projects/ tasks you consider.

Your report for **Part A** should include:

1. **Overview and Justification**: A report on the project you selected, summarizing the relevant characteristics you considered when making your selection. Beyond whatever additional information you collect in your research, include at least a name, a website link, and a brief description of the project (what it does, who uses it, etc). Explain the criteria your team used in selecting it over any others, referencing the collected information from your overview. You may contrast it to other projects you considered but rejected, if applicable (approximately 3 paragraphs, soft limit, also see Appendix A for some helpful pointers).

Once your team has settled on a project and candidate tasks, research your ideas in more detail. Read the documentation. Build and execute the source code, and try to read/understand it. You should explore the code to the point that you understand how your modification fits in the overall picture, and that you are convinced that it is both non-trivial but doable with the resources (time, team members) available.

In selecting a task, consider the functional and non-functional implications and requirements of your proposed task(s), as well as how it fits in the larger project structure. We do not want a full software requirements specification. Instead, we want lightweight documentation of your tasks' requirements and evidence that you understand how it fits into the larger project.

Your report for **Part A** should also include:

2. **Successful Build**: Evidence that you can build and run the software (e.g., a screenshot or text output from a successful build, a screenshot of the running program). Getting an open-source project to build/run can be a huge effort, and we want to mitigate this risk.
3. **Tasks Description**: A brief textual description of your proposed change(s). If you are proposing several changes, list all proposed changes and a priority order. Depending on how difficult the changes end up being, you may do not necessarily have to implement all of them. However, if your actual changes deviate from the plan, we expect a short explanation with the final submission. (< 3 paragraphs per task).
4. **Task link(s)**: Evidence that the task(s) is/are requested by the community (a screenshot or link suffices).

Once you have selected a project and candidate tasks, estimate time and effort and schedule your work. We will grade your planning, but not your accuracy after the fact. It is completely acceptable if plans change,

as long as you document the changes and their reasons and update the plan. The plan should illustrate how you will work as a team on this project and anticipate and plan for the main risks.

Your report for **Part A** should also include:

5. **Initial Time Plan**: Choose any format as long as it is clear (e.g., network diagram, Gantt diagram, plain text). This should include at least: individual tasks and milestones, with deliverables; estimated effort for each task; dependencies between tasks; and a best-effort assignment of tasks to team members. If you have evidence supporting your estimates, include it. Consider how other classes/exams might interfere with your schedule. We do not expect a full QA plan in this initial report, but be sure to schedule time for QA activities. See the final team report for more on QA. (< 2 pages)
6. **Short Risk Assessment**: Identify and briefly describe key risks in each task and discuss how you plan to mitigate those risks. (2 paragraphs, soft limit)
7. **Initial Development Process Plan**: Describe the development process your team plans to follow. This should mention quality assurance and how you plan to communicate and collaborate as well as divide and integrate work. (< 1 page, soft limit)
8. **Tasks Scope Justification**: Evidence that the tasks are of a sufficient and reasonable size and complexity for your team and for this project. Your scheduling and effort estimation, below, may be used to help justify your argument here. (2 paragraphs per task)

We strongly recommend that you interact with the us during this process to verify that the scope of your proposal seems reasonable. We strongly encourage you to do this as early as possible, and before you start investing too heavily in your plans.

Submit the **initial report** covering the 8 points listed above as a single **PDF file** per team to Ritaj or via email. Include the names of all team members on the title page and ensure the document is structured to make it easy to find the 8 points. Page limits are provided for guidance; we will not enforce them. However, if you write a response that is substantially shorter than the guidance suggests, you should reconsider whether your analysis is sufficiently deep.

Part B: Project Report (120 points, team, due December 12th, 2:00pm)

After completing and submitting your contributions, write a report about the tasks you have performed. The report will include a description of the project and its business context, a description of your tasks and their context, an explanation of deviations from your plans in **Part A** (if any), a discussion of the your contribution architecture and how it fits in the large open-source project, a discussion of the guidelines and concepts that you enforced in your task code implementation, and a discussion of your quality assurance efforts and why they were suitable.

Specifically, we expect the following sections:

1. **Selected Project**: A brief description of the open-source system to which you contributed ($1 - 2$ paragraphs). You may reuse text from **Part A**.
2. **Project Context and Business Model**: An analysis of the open-source project's context and business model. This may include a short history of the project, competing open and closed-source projects, or a discussion of the developers' motivations to build this system. Essentially, we want to know why this project exists and why it is important. (< 0.5 page, soft limit)

3. **Task Description (per task)**: A description of the tasks you have implemented, a high-level description of how you implemented them , a high-level discussion on the architectural design of each task and how it fits in the open-source project, and a discussion of the software construction guidelines and concepts you have enforced in task code implementation. $(1-2$ pages, soft limit).

4. **Submitted Artifacts (per task)**: Evidence of the code, documentation, or other artifacts you produced for the task, and evidence that you submitted them to the project. We prefer links to publicly available resources (repository, email, pull request, etc), but will accept a zip file of your artifacts with a screenshot documenting the submission.

5. **QA Strategy**: Describe which QA activities you performed and justify why you selected these QA activities over others. Describe metrics if appropriate. The justification will likely refer to relevant requirements as well as to the project's practices. $(1-2$ pages, soft limit)

6. **QA Evidence**: Evidence of your quality assurance activities described above. For example, provide source code or links to source code of tests, provide test protocols, comments or protocols from code reviews, reports from static analysis tools, links to or screenshots from a continuous integration platform, and so forth.

7. **Plan Updates**: A description and justification of deviations between your initial plans and your performed activities. Changes are expected, but they should be tracked and explained. Describe changes in scope (e.g., fewer tasks) and in the schedule and work allocation. Provide an updated schedule and note differences. Explain the causes of the changes, such as unanticipated risks. $(1-2$ pages, soft limit)

8. **(Optional)** Evidence that your changes have been accepted into the code base of the open source project in forms of links or screenshots.

Page limits are provided for guidance; we will not enforce them. Collect all parts in a **single PDF** document with clear subsections and the names of all team members and submit that file via email or Ritaj.

Part C: Presentation (45 points, team, due December 12th, 2:00pm)

The last two lectures of class are dedicated to group presentations about your open-source contributions. We will randomly assign the presentation order and put all the slide decks on a laptop. We expect all team members to take an active role in the presentation. We will provide feedback sheets for every audience member. There will be $5-10$ minutes of questions from the class following each presentation. Presentation times should not exceed 10 minutes (*hard limit*).

The goal of the presentation is primarily to teach the class about the project to which you contributed, and your experiences. You should mention your contribution (the actual tasks), but we do not expect you to include, for example, any code or diagrams from your report, unless they're helpful for supporting a point about your interactions with the project. If you have contributed multiple tasks you may want to focus the presentation on a subset. Your presentation should cover the following three topics (in any order and structure you deem appropriate):

1. **High-level Project and Task Description**: Describe the project in terms of its high-level goals and the context in which it operates. This may include a brief history and the business context, if interesting or relevant. For example, it may be interesting to note that a project was spawned from a closed-source operation, or that it competes primarily with a closed-source counterpart. Include a brief description of the task(s) you performed, such that the audience has sufficient context to understand

your explanation of your experiences, below. You should not spend more than 1/2 of the presentation on describing the project and your task(s).

2. **Project Governance and Communication**: Describe the processes and tools the project uses to coordinate among contributors. Are these processes formal or informal? Provide an explicit description (possibly with a diagram) of the acceptance process used for efforts like the task you completed. If applicable, include standards or expectations regarding software engineering activities including requirements, architecture, and quality assurance; alternatively mention that no such standards exist.

3. **Your Experiences**: Summarize your experiences (and what you learned!) interacting with this community of open-source developers, focusing on any surprising or unusual aspects of the process or interaction. Did you run into any trouble understanding, changing, or contributing to a large, pre-existing project? Were there unanticipated challenges in either implementing your change, or in getting the change submitted to and accepted by the project maintainers? Did the project collaboration process or culture help or hinder your effort in any way? Characterize any interaction you had with the team leadership and community, highlighting especially any useful/useless input you received. You may (but are not required to) also relate the experience from this project with relevant experience from your industry and academic experience on other projects.

Your summary of your experiences can be at whatever level of detail you think is interesting or informative. Given the time limit, selecting and highlighting the two or three most important or interesting observations is likely more useful than trying to be complete.

All team members should have an active role in the presentation. For effective communication, you may want to prepare slides. The first slide should include the names of all team members. It is essential that you practice the presentation beforehand.

Be aware that short presentations are more difficult than long ones. Ensure that you get to the point quickly and eliminate all information that is not essential. However, still provide sufficient context information that the key issue and your reflection are understandable for your target audience. Your primary target audience is your fellow students.

You must upload your slides as a **single PDF** document (separate from the report) to Ritaj or via email. If you choose not to use slides for the presentation, you must still submit a title slide with your group members' names and a title.

During the presentations, we will hand out feedback sheets and expect that every student is able to identify a challenge or insight and is able to ask one question. We will have a few minutes for questions after each presentation.

Grading and Evaluation

This project is worth 240 points. We will grade you based on the learning goals listed above. The project description contributes 75 points (31.25%), the report contributes 120 points (50%), and the presentation contributes 45 points (18.75%).

This project is open-ended in many ways, and we expect you to use your judgment on what is reasonable and to properly identify a project and appropriately scope the work. If you have questions contact us.

To receive full credit for the **initial report** document should include:

- A description of the factors you considered in selecting a project and set of tasks, and sufficient

information on the project you selected to inform that decision.

- A clear justification for your decision regarding project selection.
- Evidence that you have been able to successfully compile and run the code.
- Clear, but brief, descriptions of the tasks you are proposing to make, and the requirements for those tasks, and evidence that the tasks are requested by the community.
- Convincing justification as to why this is a non-trivial task and is appropriately scoped for the project. For full credit, explain why the task is well-suited for teamwork and use reasonable effort estimation to demonstrate that the tasks are scoped well for the available developer hours. One way to address this issue badly is to select one independent task per team member (don't do this).
- A realistic schedule that includes work items (or tasks), milestones, dependencies, etc, including time estimates for every work items, indicating work is going to be divided.
- A list of at least two relevant risks and corresponding mitigation strategies
- An outline of the software development process mechanisms you will adopt.

To receive full credit for the **project report**, we expect:

- A description of the project and it's context and business model
- A clear description of your task(s) and what you did to complete it/them.
- A clear description of your QA strategy and the actually performed QA steps
- A justification why your QA strategy is appropriate for the performed task in the context of the system and its requirements
- Updated planning documents with a justification explaining deviations. (We do not penalize deviations as long as they are explained).
- Evidence of the submitted code and the described QA activities
- A clear description of the guidelines and principals used to implement a reasonably high quality code. Probably, the guidelines enforced by the open-source project developer or the ones you proposed to use..

To receive full credit for the **project presentation**, we expect:

- Participation from all team members.
- Effective communication of the key issues, with sufficient context, within the time limit.
- Content addressing and demonstrating understanding of all three points listed above (High-level description; project governance and communication; your experiences and insights).
- Constructive feedback for other presentations in class.

Teamwork

This project is to be done in your assigned teams. All parts except the reflection should be done in a team context and submitted on behalf of the team. You are highly encouraged to openly discuss all issues that may arise in the process of working within the teams. If severe teamwork issues arise please contact us.

Appendix A: Selecting a Project

You may select any active open-source project in any language. Examples include:

- Apache POI (`https://poi.apache.org/`), Apache Spark (`https://spark.apache.org/`), Apache Hadoop (`http://hadoop.apache.org/`), and Apache Cassandra (`http://cassandra.apache.org/`)
- Eclipse (`http://eclipse.org`). For example, add cross-platform support for opening a file on the command line (`https://bugs.eclipse.org/bugs/show_bug.cgi?id=4922`). There are other ideas available at `http://wiki.eclipse.org/Summer_of_Code`.
- Audacity (`http://audacityteam.org/community/`)
- PostgreSQL (`https://wiki.postgresql.org/wiki/Todo`)
- Adding autosaving for untitled editors in JEdit (`https://sourceforge.net/p/jedit/feature-requests/`)

Other places to look include:

- GitHub (`https://github.com/`). Click on Explore at the top of the homepage and look through trending projects by programming language, or check out the currently most active projects (`https://github.com/trending`). Look through an interesting project's wiki and issues list to find worthwhile tasks.
- Apache projects (`http://www.apache.org/`)
- Mozilla projects (`https://developer.mozilla.org/en-US/docs/Introduction#Find_a_bug_we%27ve_identified_as_a_good_fit_for_new_contributors`).Mozilla has a number of Open Source projects (including Firefox and Thunderbird) that are actively being developed and they recommend bugs for new contributors.
- LibreOffice Easy Hacks - (`https://wiki.documentfoundation.org/Development/Easy_Hacks`)

Finally, here is a sample list of tasks and projects that have been previously done by other students. Some of the tasks listed were only part of the group's overall contribution:

- Adding single objective Grey Wolf Optimizer (GWO) algorithm to jMetal framework (`https://github.com/jMetal/jMetalCpp/pull/14`)
- Resolve Build Warnings on DNN.Platform Solution (`https://github.com/dnnsoftware/Dnn.Platform/issues/2648`)
- Compressing log files after rotation in Tomcat (`https://github.com/apache/tomcat/pull/223`)
- Server side event framework built on Java EE APIs (`https://github.com/TareqK/Jesse`)
- Adding datagrid filters instead of using the columns filters in zfc-datagrid (`https://github.com/zfc-datagrid/zfc-datagrid/pull/81`)
- Implemented find and replace in DrJava (`http://www.drjava.org/`)
- Create a test suite with 100% coverage for 2 packages and refactored a method in Lime Text (`https://github.com/limetext/lime`)

- Implemented a download after play feature, added information to the program output, and the option to convert a song to a mp3 file in mps-youtube (`https://github.com/mps-youtube/mps-youtube`), a command line youtube player
- Added the ability to handle fractional inputs to LibreOffice Draw (`https://wiki.documentfoundation.org/Development/Easy_Hacks`)
- Merged similar methods and added error handling to the create function in Elasticsearch (`https://github.com/elasticsearch/elasticsearch`)