CS 471: Software Engineering
Spring 2018
Project Assignment (Part 3 of 3) – Sprint 2

Due date: Friday, April 27, 2018 (at the end of day)

1. Description

This third part of the group project (see timeline below) asks you to **plan** for and **complete** your second and final sprint for this semester, using a similar process to the one in Sprint 1. In CS 481 we aim at completing four sprints.

The grading rubric emphasizes scrum artifacts, especially the **Sprint Backlog**, and introduces an artifact known as **Quality Plan**.

The maximum points for Sprint 2 is 250, not because it is more complex, but rather because Sprint 1 is considered a "dry-run", and hopefully in Sprint 2 you will show some improvement and better team management/distribution of work compared to Sprint 1.

Part 1 of 3	**Part 2 of 3**	**Part 3 of 3**
Product Backlog (100 Points)	Sprint 1 (100 points)	Sprint 2 (250 points)

All resources referenced in this file are found on Piazza under "Projects".
Below are the steps required to complete this project assignment.

2. Read about GitHub and ZenHub conventions

Get familiar with the CS471_GitHubAndZenHubConventions_QuickStartGuide (henceforth referred to as GitHubConventions) document containing the conventions you will need to follow for the group project, until the end of CS481. You should already have access to a private GitHub repository and you should have already installed the ZenHub browser extension (Firefox or Chrome) on your machine.

Personalized feedback based on the GitHubConventions is provided in the Scrum Linter report (shared with the team in the Google Spreadsheet file "`CS471 S18 ScrumLinterLink_ProjectGradesFeedback - <TeamName>`") and the team should fix all the inconsistencies indicated in the report.

3. Read About Metrics

Read the `CS471_S18_Sprint2_Metrics.xlsx` spreadsheet (found on Piazza) to learn what you need to track in Sprint 2 **before** you begin. Note that there are a few changes in the metrics spreadsheet from Sprint 1.

4. Quality Plan Background

A **Quality Plan** is an engineering (not a Scrum) artifact. While Scrum is focused on the development process, engineering activities deal with engineering technologies. Quality Plans have been around for decades and there are many forms of them; you may discover your employer will ignore them (with the assumption that "someone" is taking care of that) or dictate a specific outline/template for you to follow (think... aerospace, medical devices, or other life-critical software). To keep things simple in CS471, your Quality Plan will be very brief and will answer just two questions:

1. **What is your project's Quality Goal (expressed as defect density)?**

2. How will your Definition of Done achieve that goal?

With regard to **Question 1**, there are many possible goals. We use **defect density** to keep things simple. Industry projects tend to either ignore quality goals (hoping quality won't become a problem until after we get more funding:) or they reference a previously released product (e.g., "CheckMate V3 needs to be at least as good as V2"). In practice, quality goals must be appropriate for the business and customer. The flashlight app for your phone will not have the same quality goal as the space shuttle's avionics. In your CS471 project, you may assume the following defect density goals are appropriate:

- Aerospace, medical and related life-critical software (e.g., self-driving cars, automated braking systems, etc.): <0.5-1.0 defect/KLOC
- Server-side software or operating systems with thousands of *supported users* and facing the public Internet's attacks: ≈2.0 defects/KLOC
- Client-side application with thousands of *supported users*: ≈5.0 defects/KLOC
- Free mobile application with minimal support (e.g., a discussion group): ≈7.0 defects/KLOC
- Throw-away prototype code without supported users: 10...25 defects/KLOC

Notes:

- The term *supported user* refers to whether your sponsor provides online or telephone support for the product. We care about this because support costs become a liability for your sponsor's business, and are related to the number of users, the cost of answering each support call, and the defect density delivered to those users. *Throw-away* prototype code refers to code that will be discarded after the project demonstrates some achievement; because throw-away code won't be enhanced or repaired or even leveraged into another product, the development team is often focused on demonstrating features rather than long-term costs. Warning: more than one engineering team has set out to develop a throw-away prototype and, following a successful demonstration that carefully avoided known defects, heard the question, "How long will it take before we can release this to our customers?"
- The defect removal activities discussed in CS471 are unlikely, by themselves, to achieve less than 1.0 defect/KLOC because this level of quality requires techniques (e.g., proof of correctness, auto-generated tests, etc.) above and beyond our scope. *Green field* software (i.e., not leveraged from an existing, released product) developed with no other defect removal activity except Acceptance Testing may exceed 15 defects/KLOC. Pick an appropriate goal for your project using the assumptions above.

Your answer for **Question 2** explains why the quality activities you've placed in **your (updated!) Definition of Done** will achieve your quality goal. It's a quantitative explanation (e.g., use a defect removal model with some "realistic" assumptions about the efficiency of your chosen defect removal activities). You likely won't encounter this approach in an entry-level job; your employer will likely do what they did in the last project, perhaps with some changes (e.g., "Unit-level and acceptance testing were expensive and so were our customer support costs. Let's incorporate code reviews to see if they will lower our overall costs"). My agenda, again, is to prepare you to become a senior-level engineer who knows what to do when the "same old thing" won't suffice, and to create a quantitative argument that will defend your plan from your colleague's and management's critique (e.g., "Well we've never used code reviews before and they'll just slow us down even more.").

5. Create a Quality Plan Document and a Defect Removal Model

Use the `CS471_S18_Sprint2_TemplateQualityPlanAndDefectRemovalModel.xlsx` spreadsheet (found on Piazza) to create a Quality Plan that contains:

- Your project's quality goal expressed as a defect density (i.e., defects/KLOC)
- A Defect Removal Model based upon the assumptions in the `CS471_S18_Sprint2_TemplateQualityPlanAndDefectRemovalModel.xlsx` file

Revise your Definition of Done as necessary after your defect removal model asserts the chosen

activities will achieve your quality goal.

6. **Scrum Roles**

You can keep the same scrum roles, or change the roles in this sprint. The information should be updated in the README.md file (see GitHubConventions).

7. **Sponsor meetings**

Meet with your sponsor (if you have not done so already) and review the **Product Backlog**.

- Review/revise as necessary the description and status of each **User Story**.
  - Use your project's **Definition of Done** (DoD) to resolve questions about whether a story is complete. If your DoD is not working, raise this issue in your Sprint Retrospective Meeting and change it between sprints.
  - If a story is complete (per DoD) but the sponsor wants something different, create a new story (and Acceptance Criteria) to capture the requested change. Customer feedback (i.e., validation) is an important part of scrum and you should expect to modify your Product Backlog between sprints.
  - Note that you may discover missing stories, or decide that an existing story is no longer required. For example, you can assign the stories the label `abandoned`.
- Review/revise as necessary the **Acceptance Criteria** for each User Story
- Review/revise (use the *Scrum Planning Poker* process) the **estimates** for the User Stories
- Review and revise as necessary the **priority** of each story.
  - Remember: Your sponsor gets to choose the stories and their priorities; The Scrum team is responsible for their estimates.

The meeting with the sponsors should be documented in the GitHub Wiki page (see GitHubConventions).

8. **Generate Sprint Backlog**

After choosing with the help of the sponsor the stories to be implemented in the second sprint, create 3-10 tasks for each story.

- The tasks should reference in their description the original story (e.g., #123)
- The tasks should be written in engineering language
- A team member should volunteer for a task by assigning the task to them and providing an estimate
  - NB: the developer who signs up for the task gets to estimate it
- "Research" tasks should be labelled accordingly (see GitHubConventions).
- It is normal at the beginning of the sprint to identify only ~70% of the tasks for the sprint, and as the sprint progresses the team adds more tasks (linked to stories).

9. **Working on the Sprint Backlog**

To indicate the fact that the task is being currently worked on, drag the task to the `In Progress` pipeline (see GitHubConventions).

Every commit should be linked to a task (see GitHubConventions).

10. **Submission**

One team member should submit via Blackboard a zip archive titled `<TeamName>_Sprint2.zip` containing the following files:

- `Sprint2Metrics.xlsx`: Use the template file `CS471_S18_Sprint2_Metrics.xlsx` to report your progress.
  - The metrics file requires you to provide a link to a video demonstrating the functionality of every story completed. In other words, it contains the feature

"demo" that the sponsor will see at the end of the sprint.

- `Sprint2QualityPlanAndDefectRemovalModel.xlsx`: Use the template file `CS471_S18_Sprint2_TemplateQualityPlanAndDefectRemovalModel.xlsx`.

All the remaining artifacts generated by the team (e.g., README.md, DoD, user stories, tasks, commits, etc., mentioned in the GitHubConventions) in Sprint 2, most of which would have been validated by the Scrum Linter, will be considered as the team "submission". And remember:

If it's not on GitHub, it does not exist, and you will not get credit for it.

11. Grading Rubric

The maximum points for this second sprint is 250, and the points are distributed as follows:

Product Backlog (10 points)	**Points**
Sufficient number of detailed, prioritized and estimated user stories (in role-goal-benefit, and written in customer's business language) having adequate acceptance criteria in the Given-When-Then template	10

Sprint Backlog - Stories (35 points)	**Points**
User stories (in role-goal-benefit) proposed for sprint	10
Proposed Stories Prioritized and Estimated	10
Adequate Acceptance Criteria (in given-when-then template) for user stories	15

Sprint Backlog - Tasks (65 points)	**Points**
Tasks defined and linked to each User Story	25
Tasks have engineers and estimates	25
Evenly distributed workload among team members	15

Quality Plan and Defect Removal Model (40 points)	**Points**
Proposed Quality Goal of Project (expressed as defect density)	5
Explanation of your Quality Plan	15
Defect Removal Model for chosen Quality Plan	20

Other artifacts (100 points)	**Points**
(Updated) Definition of Done that reflects the Defect Removal Model	15
Complete traceability between stories, tasks, commits and pull-requests (i.e., using #ID)	30
"Constant" work speed throughout the sprint (illustrated in the burndown chart)	15
Metrics and demo videos of your sprint 2 user stories passing the AC (see metrics file)	40

Individual Penalties:

- Team member was "absent" from the sprint, did not participate or got involved, or had limited contributions (up to -100% of grade)

Team Penalties:

- Team did not meet with the sponsor in the first week of the sprint for the Sprint Planning meeting (-50% of grade)
- Team did not meet with the sponsor in the last week of the sprint for the Sprint Review meeting (-50% of grade)
- Team did not use the GitHubConventions or team did not fix all the inconsistencies indicated in the Scrum Linter report (up to -100% of the grade)

Individual Penalties:

- Team member was
  - "absent" from the sprint,
  - did not participate,
  - did not contribute or
  - did not get involved (up to -100% of grade)