CS5500 - FOUNDATIONS OF SOFTWARE ENGINEERING

TEAM-5 REPORT

Summary of Project goals.

Team 5's project goals can be separated into two overarching goals. The first is to use this project as an experience to learn and practice best methods for Agile software development projects. The second was to deliver a product that captures the core requirements of the Project Product Backlog. To achieve both of these goals, the team had to carefully plan to optimize for the particular backgrounds of the team and to take full advantage of the limited time available in the semester.

The team consisted of four members: two members working through the ALIGN program, one who comes from the Mechanical Engineering department, and one who has some industry experience without proper software development life cycle methodologies. Since the team lacked proper software development processes experience, the team targeted the following subgoals:

- Develop experience with key software development tools for Scrum and Continuous Integration (Jira, Slack, Scrum, Jenkins, Junit, Git)
- Practice proper Sprint cycles (reviews, planning, Scrum meetings)
- Develop good team communication/dynamics
- Experience different roles within the Scrum process (Developer and Scrum Master)

The majority of the team did not have experience working on complex product backlogs such as the one mentioned in our second goal. Thus the team leveraged the guidance of the teaching assistant and the professor to help develop reasonable Sprint goals to tackle the project backlog each Sprint cycle. These were the goals for each respective Sprint cycle:

1. Creation of initial foundation for development including: creation of system class diagram, review and setup of initial code base, unit testing of initial code base, and deployment capability to a cloud-based platform.

2. Creation of initial foundation for user interactions including: direct messaging, message/user persistence, and user login/creation via GUI.
3. Complete One-to-one messaging, groups operational w/ moderators, and get database up and running and storing all relevant data.
4. Targeting a Demo by connecting backend features to GUI, adding additional messaging capabilities, and material preparation.

Overview of Results

At project completion team 5 successfully coded and implemented 33 User Stories/Tasks per Jira. Completed issues started with Jira issues 1 through 6 in Sprint 1 where the initial code was evaluated and planning started in earnest. Subsequently in Sprint 2, the only issues completed were 106 & 113 that were front-end centered and were centered around establishing a usable front end and for users to be able to login. Sprint 3 saw significantly more progress with implementing issues 8-10, and then 24-31 all around messaging and groups. Also, 45, 49, and 105 facilitated a persistence between instances. Sprint 4 came to a close with sentiment polarity implemented (85/86), a basic friend implementation (114), preparation for the final presentation (124 - 127), significant front end work (116, 117, 122), resolution of our testing/timeout issues allowing the team to push code to master (128), and finally a minor change to the implementation of the group builder (129).

Of the items in the backlog, not all issues that were attempted were able to be implemented by the end of the project. Issue 88 giving the user control of sentiment analysis as well as 118-121 expanding the front end to match all back end implementations could not be completed in time.

After the construction of the SRS we translated the requirements into a user story for each so the backlog was significant from the start of the project. Issues not tackled in the development process include, advanced friend features (seeing mutual friends), twitter like ‘following’ of other users, instance heartbeat/service crash notification, throughput measurements, security/passwords/login, multiple language support, multiple message type support, subpoena behavior, content filters, advanced messaging (‘private’, ‘self-destructive’, ‘forwarding’ etc), meetings, health tracker. Some issues were discussed and planned, but never made it into a Sprint. Such partially developed issues include, text translation, user icons, users control statuses (do not disturb/away designations etc), count of messages/users, hashtags. In all, the backlog over the course of the project was approximately 30% completed, but the remaining 70% were largely advanced features/extras while the completed issues were essential to the project. (See figure 1.1 Cumulative Flow Diagram)

The code at the completion of the project saw no bugs, no vulnerabilities, no technical debt, no code smells, no code duplications per Sonarqube (see figures 1.2 & 1.4). With 116 total unit tests the team accomplished 81% overall code coverage. 97.0% line coverage and 53.6% condition coverage (see figure 1.3). The code met all quality metrics set out at the start of the project. The areas of code lacking coverage are largely the database method calls due to a significant number of conditions to reach SQLExceptions and the difficulty of recreating a SQLException within. Overall the project quality is considered satisfactory.

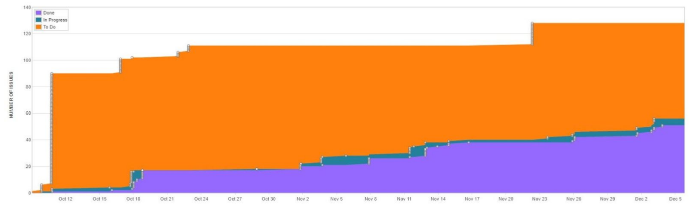

Fig. 1.1 Project Cumulative Flow Diagram

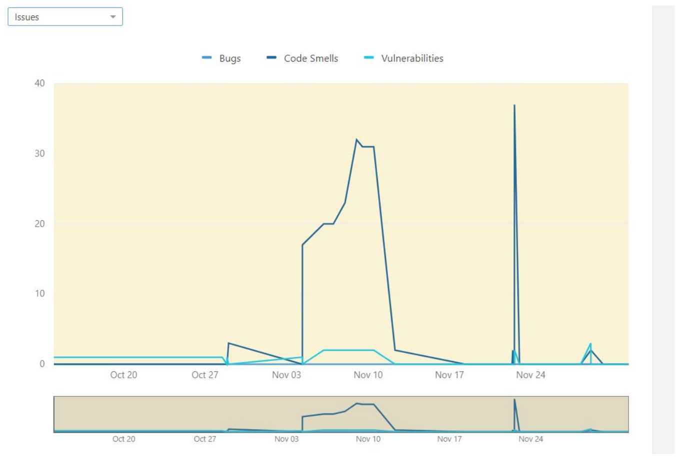

Fig. 1.2 Sonarqube 'Issues' chart. Note: the spike in issues late in the project was due to sonarqube briefly reading an old version of the code which was immediately fixed

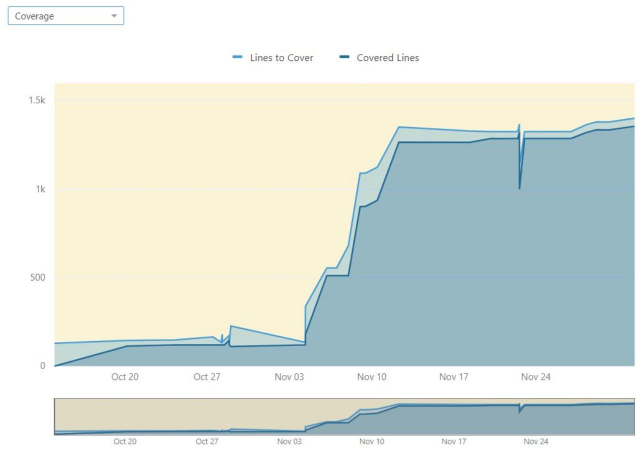

Fig. 1.3 Sonarqube Coverage Graph

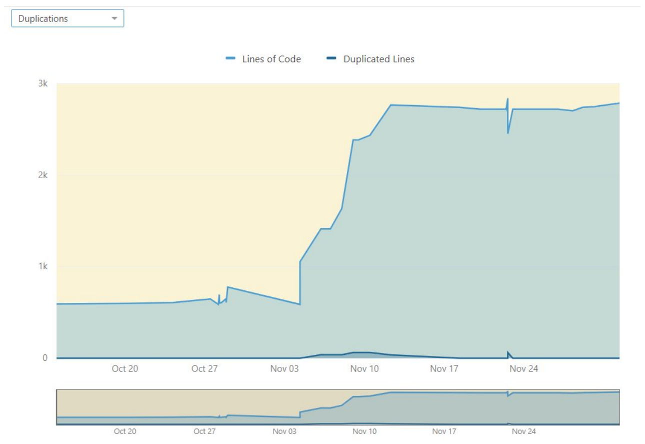

Fig. 1.4 Sonarqube Duplications Graph

Development Process

While there are many software development life cycle approaches, for this particular project, the team utilized an Agile development approach. While many other techniques are still in place and have their advantages, Agile is growing to become one of the most popular SDLCs due to its effectiveness, flexibility, and success rate (Fig 1.5)

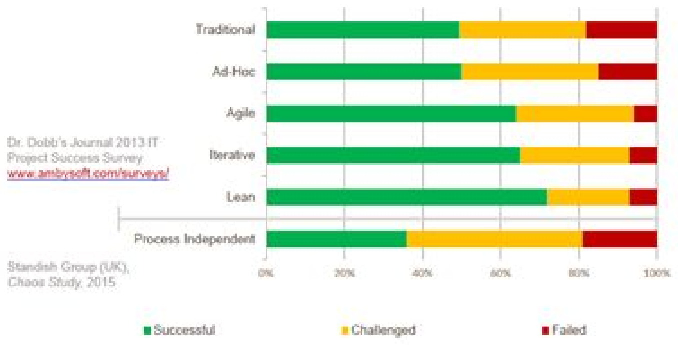

Fig. 1.5 SDLC Comparison Chart

In developing under the Agile model, we also utilized the Scrum framework which is one of the implementations of the Agile methodology. The Scrum model allowed us to iterate quickly, deliver results on a regular basis, and gave us the opportunity to self-organize which is an important value in the Scrum model.

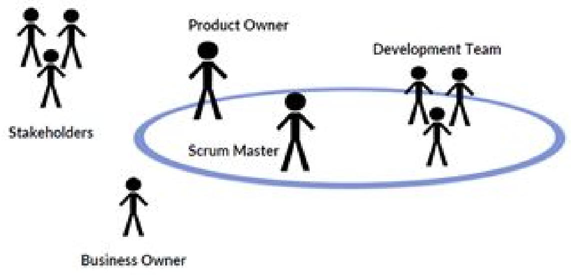

Fig. 1.6 Scrum Team Structure

By utilizing the Scrum framework, we incorporated the Scrum team structure as shown in Fig 1.6. For each 2-week sprint, we allowed a new team member to take on the role of Scrum Master. This is a critical role in the development process because this person takes special care to eliminate roadblocks the development team is facing, hosts daily standups, and takes ownership of the sprint goals and product backlog. We believe this approach worked quite well as it gave us all an opportunity to be the Scrum Master which is a much different role than being a member of the development team.

In Scrum, sprints tend to be anywhere from one to four weeks. For this project, we had 2-week sprints. We believe that having 2-week sprints was a good sprint duration given the project timeline and our team's availability. If it were any shorter, it would have felt rushed and we would have probably wasted too much time preparing for the sprint reviews. If it were any longer, we would have gone off on bigger tangents and gotten off track. Two weeks gave us the right amount of time to plan, develop, and regroup for the sprint reviews. On that note, we also believe the sprint reviews worked quite well as it was an opportunity to get feedback and direction. Vaibhov was an excellent resource for planning and prioritizing.

While many aspects of the development process went well, there were a few things that could use improvement, both for our team specifically, but also for the course. One of the inevitable challenges is that we all have other high priority things going on. Steve, Alex, and Chad all work full-time and Dayton is a full-time student. We all also live pretty far from each other, making it difficult to meet regularly. After a few weeks though, we worked out a good system. We met right after class on Tuesdays while we were all in the same room. We utilized Slack for daily communication and stand ups. We also used Microsoft Teams as a way to have remote meetings where we could screen share and walk through programming, SRS, or UML challenges. It wasn't as good as meeting in person but it worked well for our team.

Another thing that was challenging was having sprint reviews on Friday nights. I think everyone would agree that Friday nights and Saturday mornings were not ideal times to have these reviews since many people travel on the weekends. My recommendation would be to move these to weekdays.

From experience, daily standups are best when done in person. However, this was not possible. Therefore, we utilized Standup Alice, a plugin for Slack. While this was better than not having anything, there are probably better ways to conduct standups which would include feedback from team members instead of just saying what you're working on.

Overall, it was a great learning experience to work with the Agile development structure and specifically Scrum. It taught us all a bit more about the formal development process versus simply writing code, which is what we are probably most used to.

Retrospective

The project, long and arduous, was definitely a learning experience. By the end of it, we definitely learned to appreciate the scale and scope of the project versus the time allotted; we packed the SRS with the knowledge that we likely wouldn't get to everything, but even then didn't reach nearly everything in the backlog we expected.

Over the course of the semester, we learned a lot about team management, especially in terms of prioritizing a subset of tasks and focusing on those. Working remotely with others on tight schedules and with differing levels of experience also posed a unique challenge we had to overcome time and again. We became very familiar with Agile methodology and with the importance of scrum—the difference in our productivity levels between high and low-scrum sprints was starkly apparent.

On a more technical level, members of the team individually learned a lot about integrating the backend with different parts of the overall system (e.g. Chad with the GUI, Steve with the database). There was definitely a significant learning curve with these and the technologies required for the course and for completing the project.

In the future, the course could potentially benefit from more support with the required technologies (Maven, prattle, Jenkins/git [master pushing problems for half the sprints]). Having explicit information at the start of the course about the necessary/expected time commitment for traversing these learning curves for students of different experience levels would definitely be helpful. Also, providing a rough estimate of when certain key features should be expected to be implemented over the course of the semester to stay on-track would also be helpful. Similarly, earlier discussion with TAs/prof about where to focus first would be appreciated; we were kind of off-base and got behind until somewhere around sprint 3.

Overall though, the project definitely gave us an important teamwork experience and was a great exercise in software development.