PROJECT MANAGEMENT SPECIFICATIONS

for

NAVATAR

Version 1.0 approved

Prepared by Matthew Berger, Liam Gomez, and Connor Parkinson

Advised by Professor Eelke of the UNR CSE Department

February 15, 2017

Contents

1 Abstract

Navatar is an indoor navigation system for visually impaired students. Unlike existing systems that rely on expensive equipment, Navatar only requires an Android smartphone. The phone's GPS, accelerometer, and compass are used to approximate a user's location and movements and the phone's speaker is used to provide auditory directions to rooms within a building. Buildings on University campuses often have the GIS maps and the design required for Navatar's accelerometer and compass based localization, which is why students are the intended audience for the project. With the application's open source status and modular design, crowdsourcing efforts can be used to integrate maps for buildings on any campus. The application is still under development and Team 6 is working on project management and scheduling for the implementation of a variety of new features alongside other open source contributors.

2 Project Updates and Changes

Team 6 has been working on cleaning up the existing code base to better facilitate the implementation of the planned features while also debugging issues encountered with the existing code. The team has recently added additional priority to fixing bugs with the existing maps and map creation functionality as their advisor, Eelke Folmer, informed them a university has expressed interest in the development of Navatar. Efforts are also being made to automate setup of the development environment to allow other open source contributors to easily assist with improving Navatar. It became evident after the design and prototype stage that some of the team's ideas were better in theory than in implementation. One such example was the gesture-detection as a form of interaction with the application. Gestures that are context-aware lead to the same application interactions eliciting separate and highly distinct behaviors that may lead to unexpected results for the user. The context of the application may be difficult to discern for a blind user. In this sense, gesture-detection should be superseded and brought to obsolescence by voice commands. We changed our design to use Google voice to implement the actual voice recognition and voice commands. Simply speaking to the application through a microphone on a headset or set of ear-buds will allow for more accurate control and potentially limitless interactions with the app that gestures simply would not allow for. There are only a limited number of gestures, and the overlap with existing accessibility applications would make the usability ineffective. Voice commands are the chosen options here. After designing, prototyping, and testing the GPS features we found that reliably detecting a user's location indoors was difficult. It was determined that the real use-case for autolocation would be when a user enters a building, which allows for the best location accuracy using GPS. The subsequent use cases including the user leaving the building or navigating between rooms inside the building are made possible using an Android feature that allows for getting the last known location of the smartphone.The autolocation functionality is done using Geo-Fencing currently, and can be improved by using additional location data sources such as WiFi or cellular towers. As for navigation history, it is a nearly complete and functional feature barring the additional planned functionalities of allowing users to reverse the previously navigated route and collecting data on which routes are completable. Additional focus for improving the feature is allowing users to navigate the selections with easily spoken voice commands. Overall there is much to work on and many features to develop with the overarching task of refactoring the codebase to improve the quality of the existing code.

3 Project Deliverables

This team's primary focus is implementing new features that enhance the accuracy and usability of the existing application. Each team member will be responsible for the development and implementation of their respective component outlined below.

3.1 Route History - Liam Gomez

This component is responsible for logging and storing a user's previously taken routes. Storing the previous taken routes allows for a route repeat feature to be implemented, where a user has the option to view their route history and start a new navigation from that history. Storing route history also allows for a user to a reverse a given navigation after reaching their destination.

3.1.1 User Stories

1. As a user, after opening Navatar, I should be given an option that takes me to the route history page so I can view my route history.
2. As a user, on the route history page, I should be able to select a route from my history so I can repeat that navigation.
3. As a user, after successfully completing a navigation, I should be prompted if I would like to reverse that route so I can navigate back to my origin.
4. As a user, on the route history page, I would like to avoid having failed or incomplete navigations saved to my history so I can avoid repeating them.
5. As a researcher, I should have access to the route history logs in a parsable format for analytical purposes.

3.2 Geofencing - Connor Parkinson

This component is responsible for the automatic selection of the campus and building based on the user's current location. This component will use a combination of the smartphones GPS, cellular, and Wi-Fi to provide locational data. This locational data will then be compared with the known geographical boundaries of available campuses and buildings.

3.2.1 User Stories

1. As a user, I want Geofencing so that I don't have to select the campus I am on.
2. As a user, I want Geofencing functionalities to be active when I open the app.
3. As a user, I want to revert to existing campus and or building selection if Geofencing fails (is not accurate enough or times out).
4. As a user, I want Geofencing so that I don't have to select the building I want to navigate within.
5. As a user, I want Geofencing to select at least the campus even when accuracy is not high enough to select the building.

3.3 Gesture-Based Controls - Matthew Berger

This component is responsible for providing an additional input method to the user that is outside the existing Google Talkback integration. For a user with visual impairments navigating content heavy menus and input fields can be quite tedious. To improve this, a gesture based input system will be implemented and integrated with the existing user interface. Gesture based input will allow the user to make gestures on their devices screen that control the applications behavior.

3.3.1 User Stories

1. As a user, I want to be able to control the application without having to speak in noisy environments.
2. As a user, I want gestures to be consistent, intuitive, and reliable.
3. As a user, I want to be able to long press to add a landmark.
4. As a user, I want gestures to be available for the entire screen, not a subsection.
5. As a user, I want to be able to swipe to confirm or reject application prompts.

4 Project Assignments and Schedule

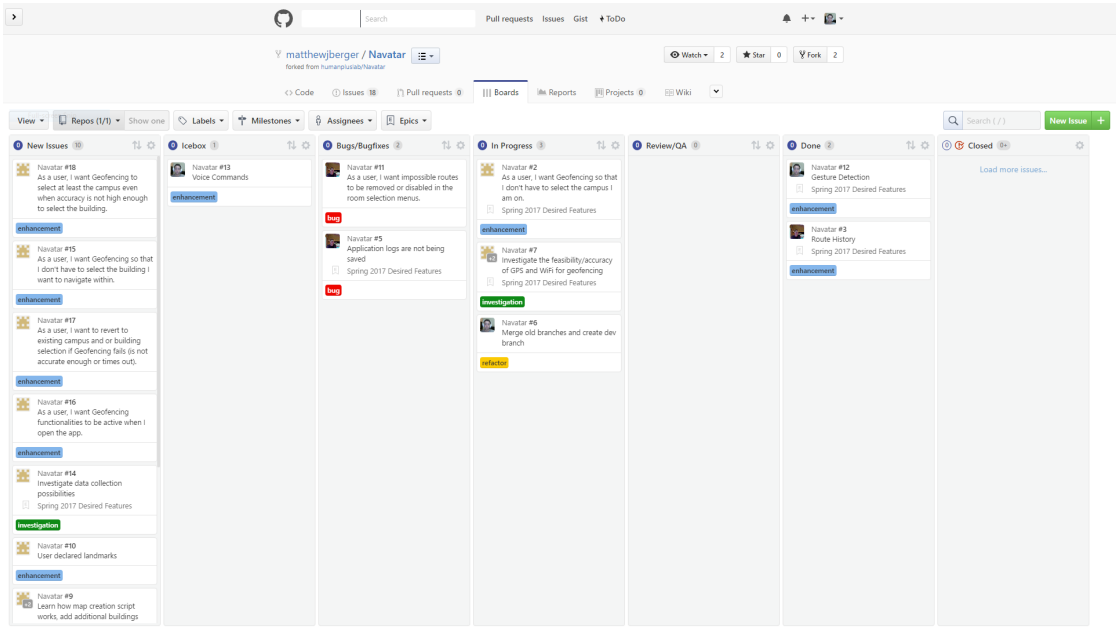
Our group is using the ZenHub chrome extension in combination with Github's issue tracker for project management. Using this tool, it has been very easy to assign work items to the members of group as well as manage collections of similar features and categorize desired and existing features. As of now, all of the major work items listed under the 'Spring 2017 Desired Features' epic on the Github repository's issue tracker can be feasibly achieved by the end of the semester on May 5th. All other features listed on the tracker are there for future reference but are not vital to the success of the project as it stands this semester. Navatar is a free and open-source software project, therefore it is important to make a note of any and all issues as they come up. We have specifically selected the most important issues to be included in our epic for the semester.

Click here to view the Navatar repository on Github.

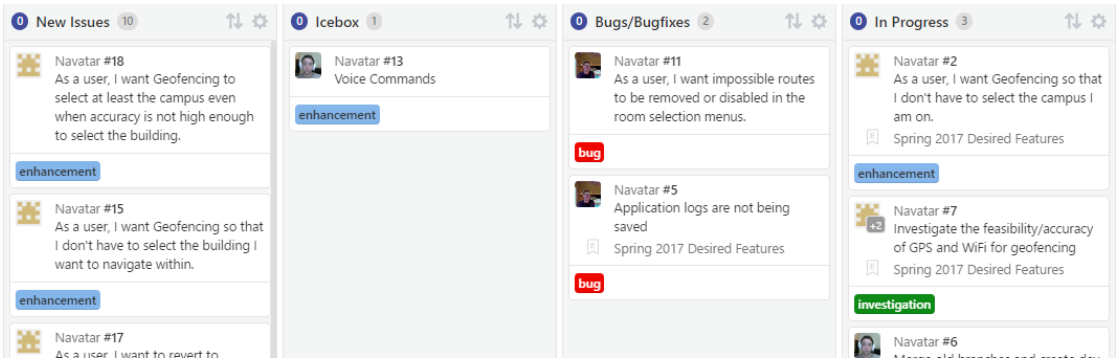
4.1 The Zenhub Scrum Board

With the ZenHub chrome extension installed, the Github issue tracker gets an extended set of features more suited to scrum project management. One such feature is a scrum board that's directly linked to Github issues on your github repositories. In our case, the issues we've created for Navatar can be categorized and managed on the scrum board through the 'board' tab that's added along with ZenHub.

Click here to view the full scrum board on Github (if you have the ZenHub chrome extension installed).



(a) The Full Scrum Board



(b) A Close-Up of the Scrum Board

Figure 4.1: Scrum Board Screenshots

4.2 Github Issue Tracking with the Zenhub Extension

4.2.1 Epic Issues

One feature that's added with the extension is Epic issue management. Epic issues are collections of issues to be prioritized and completed, and any issues created after an epic is made have the option to be assigned to an existing epic.

Click here to view the full issue tracker on Github.

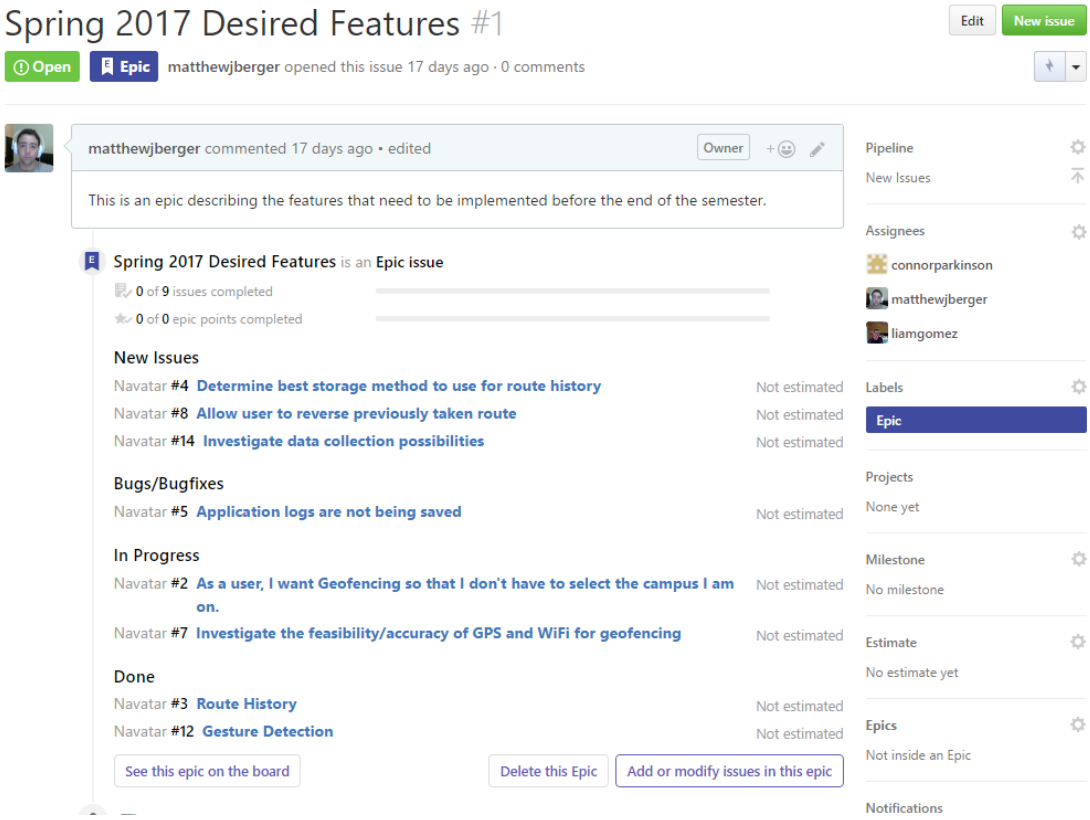
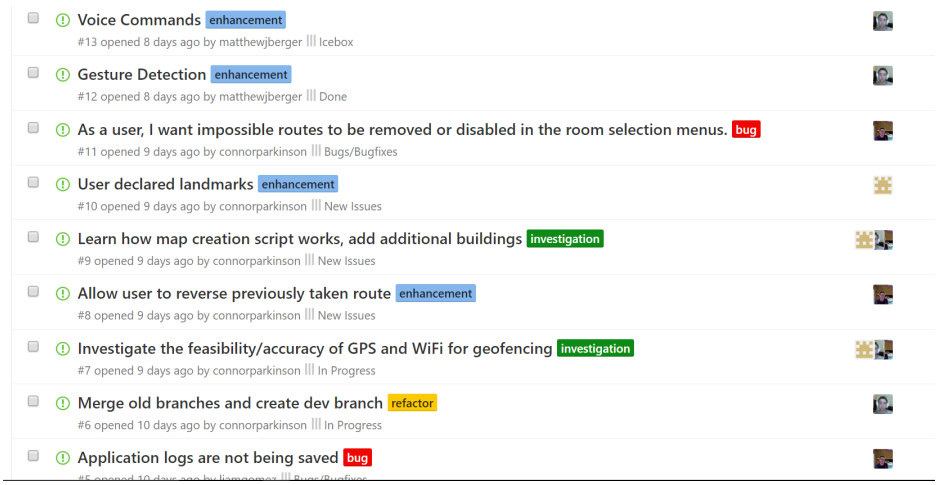


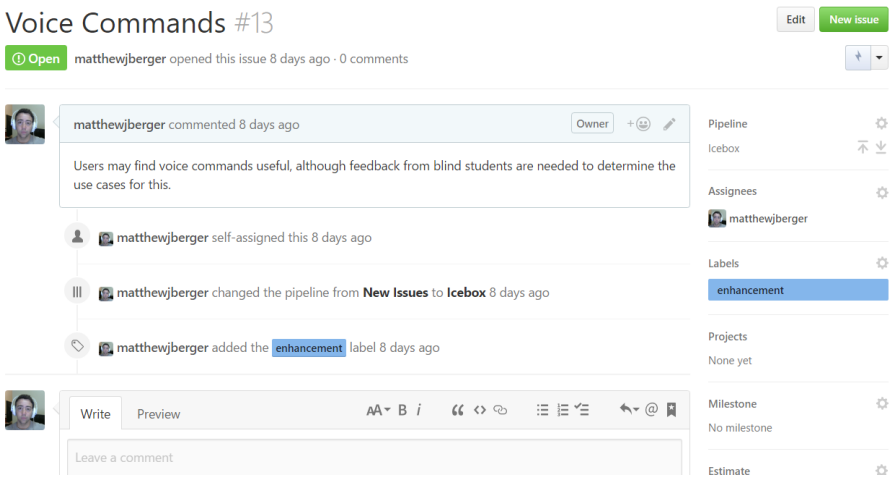
Figure 4.2: An epic issue.

4.2.2 Issue Categorization

Issues on Github can be labeled and assigned authors in order to make finding and identifying issues more accessible. With Github's issue tracker, all of the issues listed are searchable by tag, name, author, and date through Github's built-in search capabilities. Each issue contains a conversation about the issue, and can reference other issues, commits, repositories, etc as well be resolved or closed categorically by a collaborator on the repo. These changes are all automatically reflected on the scrum board via the ZenHub chrome extension.



(a) Labeled issues on the board.



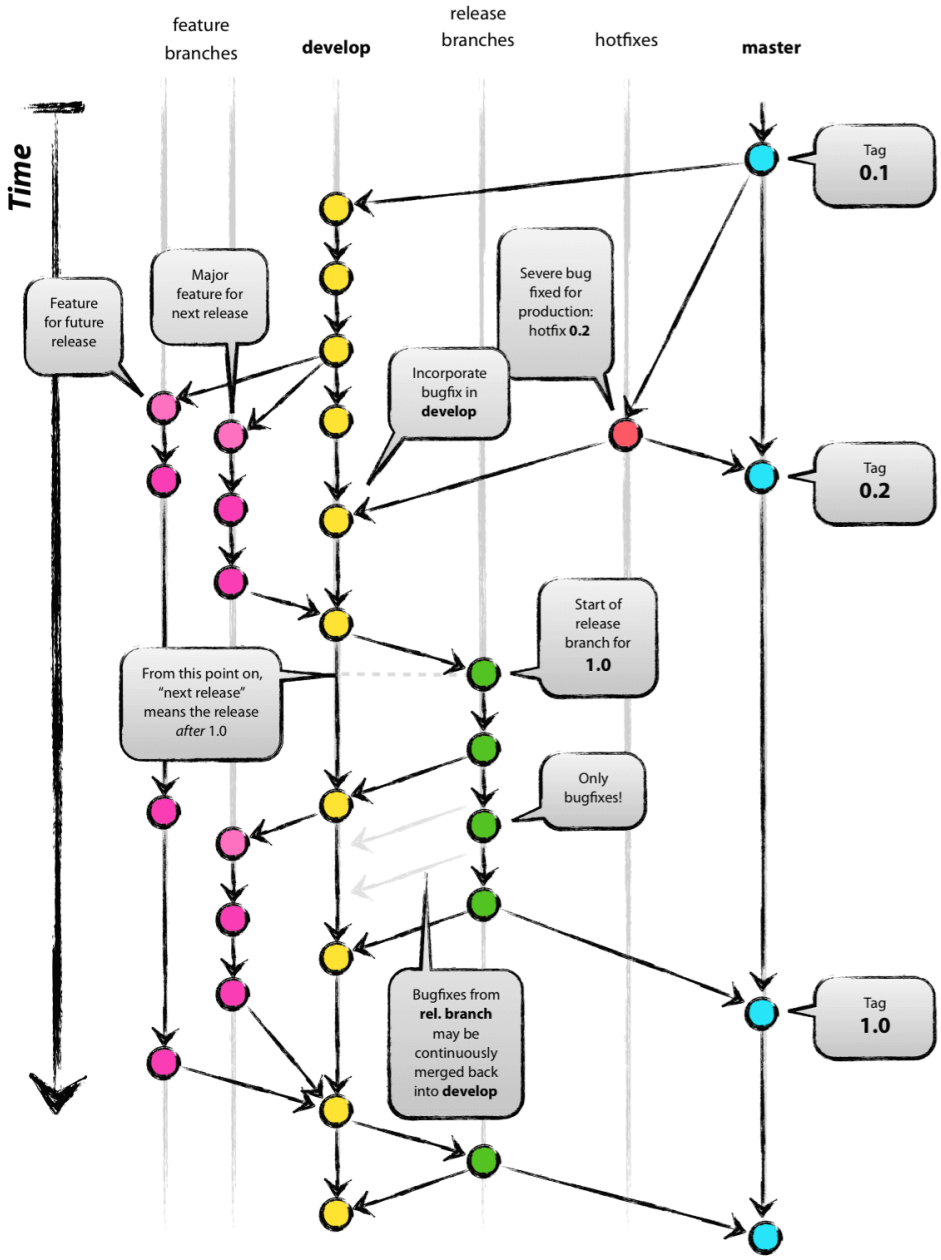
(b) A Navatar issue discussion on Github.

Figure 4.3: Scrum Board Screenshots

4.3 Branching Model

A successful git branching model is vital for organized but quick development of a project. In our project, we plan to follow the github branching model detailed below.

Click here to view the article discussing this branching model.



5 Project Monitoring and Risk

Following the updates to the requirements and design of the planned features, the team developed a schedule and timeline to ensure project success. The team delegated subsystems, responsibilities, and workload by having each team member volunteer for pre-established subsections and responsibilities based on their experience and interests. Project tasks were outlined on the scrum board with corresponding time estimates and the team developed the schedule with both realistic and stretch goals for what is planned to be accomplished. Weekly meetings have been established which allows for all team members to hold each other accountable for their assigned tasks and for the team to re-allocate time for tasks which could be deemed more complex than originally anticipated. A Gantt chart was considered for outlining each scheduled task and dependencies, but the team deemed it unnecessary due to the independent design of each planned feature.

5.1 Risk Management Strategies

Sudden Increase in Requirements or Development time

1. Mitigating the risk of sudden requirement or time increases has been mitigated by thoroughly establishing the requirements and design with a corresponding schedule that allows for flexible time allocation.

Productivity Issues

2. By having a realistic schedule and frequent team meetings, the team members are held accountable for the progress expected to be achieved at every step of the development process.

Scheduling Issues

3. Establishing frequent team meetings is possible by having all team members commit early in the development process to certain times and dates with sufficient time to prioritize and plan.

Bugs with the Existing Codebase

4. The team understands that issues can arise when working with existing code which is why the planned features have been designed to be as modular as possible to avoid conflicts with existing code. The team has also allotted time for debugging essential functionalities.

Development Environment Consistency

5. Developing an application for Android involves a multitude of version specific dependencies which can be difficult to maintain and resolve conflicts between. The team is using Vagrant to configure a standard which is being used by all team members and will be suggested for all other open source contributors.

Unit Test Coverage

6. The application has a lot of existing code, and such code may or may not be guaranteed to work. To ameliorate this and provide a safe way forward, we intend to write tests for portions of the code that we touch in order to preserve the functionality in place and assure that new functionality is robust and well-integrated.

Scope Creep

7. There are many modular features planned, but approaching too many at once can lead to a large increase in the amount of time per task and ultimately prevent the app from seeing completion by the end of the semester. To prevent this, we are planning out the project carefully and assigning specific, isolated, and modular features to members of the group and allowing an ample amount of time to complete and integrate them. This will help to prevent scope creep, and assure functionality is added to the software by the deadline.

Development Conflicts

8. To properly manage the project on github and ensure each developer can work on the project properly and without breaking the codebase, we will follow the git branching model listed in this document and be deliberate in our usage of branches and commits. All commits will contain a full and accurate commit message. Additionally, continous integration will be used to run unit tests on each commit so at a glance it will be known whether or not a prior commit broke the build and we can address the issue with expedience.

Team Coordination

9. Without proper communication, the team will not be able to coordinate their activities and develop properly. The simple solution we have come up with is to use sms, google hangouts, and shared google documents to properly interact without each other for our project necessities.

Team Motivation

10. To prevent the team from becoming overwhelmed by deadlines and features, milestones will be set and will contain small goals that comprise the larger issues. This will assure constant progress and will keep the team motivated to continue working on the project. Additionally, to motivate others to work on the project we will make the repository and codebase more friendly by adding contribution guidelines and explaining how to build the source code with the development environment provided by our vagrant box. Having other people open pull requests and submit issues will help develop the project and boost team morale.

6 Team Contributions

6.1 Matthew Berger

- Created LaTeX document
- Setup source and development environment controls
- Completed project assignments and schedule
- Created gesture subsystem user stories
- Completed five of the ten risk management strategies

6.2 Connor Parkinson

- Completed the abstract
- Outlined the project design and requirement updates
- Established the project monitoring plan
- Completed five of the ten risk management strategies

6.3 Liam Gomez

- Outlined the project deliverables
- Organized user stories on the team's ZenBoard
- Compiled the references.

6.4 All Team Members

- Peer review, editing, and formalization of this document and the knowledge contained herein.