Team 3: SWENG 500
Software Engineering Studio Final Report

Table of Contents:

Introduction

The engineering studio for the Software Engineering program provides the opportunity to develop a self-created and directed project for each team. Confronted with the task of developing a unique and complex software system, Team 3 scoured open source forums and software request pages, attempting to find a need for a system that suited the teams abilities and would provide an enjoyable challenge to overcome. After discussing background and preferred development languages and tools, we brainstormed ideas ranging from inventory applications, gaming applications, fantasy football managers, and expense report compilers. After narrowing down the ideas we came up with, the team settled on a travel-oriented application to provide team coordination for work trips. This was an application that the team could relate to and would provide a broad range of requirements that would challenge the team.

The Away Team project allowed each team member to take their knowledge and push it beyond the current level of application. The challenges of working with a team, creating a multi-faceted system where components would require integration via network communication, short timeline, and the purpose of providing a complex interaction between multiple users ensured that creative problem solving and resourceful teamwork would be required to complete the assignment. During the consideration of the project solutions, the team assessed their current skills in order to match up team members against the challenge appropriately. Peter Karski expressed comfort in several languages, but most recently worked in Java and Python, he had written several simple apps for the Android framework prior to this project, but had no experience networking applications. Stephen Naimoli began the project with a Computer Science background and comfort in a range of languages with experience in servers and integration. Clay Parker offered strong Java knowledge and experience developing web applications. Finally, David Vu also had a Computer Science background with multiple languages under his belt, which included C++, Java, Python, PHP, and JavaScript, and expressed interest in building Java UI, web development, and Java-Database integration.

Project Goals

During the brainstorming session to develop a project idea, several team members expressed interest in mobile apps and travel tools. This led to the eventual development of the Away Team concept that would provide coordination tools for sharing information between people on a work team. The idea was conceived to develop an easy way for teams, especially those on the road, to easily find contact information and schedule meetings and other events.

The team outlined the key components of the design, including the need for users to manage multiple teams, and identified the kinds of information we felt teams would want to share, settling on calendar events, group tasks, and contact information, and providing a team the ability to create expense reports on the road. Since we wanted to focus on a mobile user, the team decided to make map and location an integral requirement, which led to the decision to

develop an Android application due to the team's background and the tools the Android system offers for geolocation.

The initial requirements for the system came together very quickly as the collection of collaboration tools the team felt a mobile professional would need were well understood. The functionality of managing several teams turned out to be surprisingly nuanced and difficult to describe, so a lot of early development time was spent deciding how teams would be accessed, interacted with, and what information would be shared. Sharing usage scenarios helped identify much of the final requirements. The concise requirements that drove the project goals were:

- Sharing team member contacts
- Tracking team member locations
- Sharing schedules
- Setting group tasks
- Tracking Travel Expenses

Project Plan

An open-ended project where the team must both design the requirements and scope of the project and develop a plan of execution leads to a great deal of ambiguity. To handle this, the team started out focusing on design requirements and identifying how the system's implementation structure would be designed. As this was clarified, the team began prioritizing tasks that would allow us to reach the final goal on timeline by implementing functionality in an orderly fashion that could be built up, starting from initial interface design and display, network integration, and authentication toward the complex team management interactions and manipulation of shared data.

System Structure Design

It was apparent to the team on the conception of the project that the system would require several components. Tracking a number of teams would require a place for data to be stored and accessed utilizing authorization-based interactions. To complete this, a database would be required. The concept of the project was based around a mobile user, therefore, a mobile application was a necessity. Based on the team's background, range of application form factors, and the geolocation tools of the platform, Android was selected as the target for the mobile client. In order to completely cover all other web-enabled platforms, a web client was decided upon that would be accessible from traditional computer browsers or mobile browsers on phones that were incompatible or did not desire to use the app. Finally, in order for the mobile clients to interact with the database, a server would be required, and the team opted to implement REST compliant architecture on the server.

Task Assignments

With four designed entities and four team members, it made sense to task each system structure to an individual team member. This worked for the majority of the project, allowing each segment of the system to have an expert who could be looked to for deliverable

accountability. For the most part, this tasking worked out well, allowing team members to focus on their area of comfort, while pushing beyond their current experience and working with a team to generate an entire system rather than standalone components.

Team members volunteered for the roles as their background permitted, therefore, David took the database, Stephen offered to use his server experience to implement the server and design RESTful APIS, Peter used his Android development experience to start a mobile app, and Clay's web development was put to task designing the browser client. As the database was completed earlier than other portions of the project, due to its role as the foundation of the data structure, David was used to augment efforts in other parts of the system, usually contributing to the buildup of the API functions that were required to support the mobile and web clients.

Plan Execution

As previously noted, the team prioritized tasks early on until the current functionality progression and integration requirements began to take precedence. The major tasks that were tackled by the team included:

- Design Database
- Design and Deploy Server
- Integrate Database Schema
- Implement REST Framework
- Design and code external REST API
- Implement Android App
- Implement Web Client

For the initial few weeks, keeping a good pace toward achievable immediate goals was important to ensure focus was not lost and measurable progress was definitely being made. Therefore, for the initial portion, the team developed an ordered list of short term goals that could be used to direct teamwork and effort:

1. Finalize requirements
2. Determine specific technology to be used for each component
3. Set up basic operational baseline for each component (running server/web and app API/instantiated database)
4. Pass data using basic APIs between all components (demonstrate push/pull as appropriate)
5. Collect/store/display useful data (first function)
6. Continue to add functionality

The development of initial goals helped prevent the team from locking into design paralysis and ensured that component integration could occur as quickly as possible because the focus for each subsystem would be coordinated in a loose way to promote the accomplishment of baseline connectivity early on before trying to build up structures and advanced functionality.

Collaboration & Use of Collaboration Tools

Working with a team to design a complex system with self-determined, often vague or shifting requirements is always a challenge. Accomplishing this while being geographically separated, with limited scheduling ability and scarcity of time to coordinate meant that the use of collaboration tools and the proper selection of these tools would be imperative to the success of the team in the project. The team had to deal with work schedules and time zone differences, which placed a premium on asynchronous communication tools that would allow team members to contribute when they were available. Real Time tools, such as Google Hangouts, helped share ideas rapidly and clearly, but these were used significantly less often.

Early in the project, the team used the Angel forum, which continued to be a source for communication throughout the project, but was supplemented by important tools, the use of which contributed heavily to the ability of the team to successfully meet goals and coordinate development and integration.

Github

The team needed a tool to allow the concurrent updating of code and contributions to shared code bases. Github was selected as a repository tool that would allow this. Using github, team members were able to synchronize code updates and create code branches to pursue individual development. It also ensured that all the most current code was available to everyone on the team so that we could review each others work or reference source code to check integration solutions. Github also allowed us to synchronize any class diagrams and database schema changes. As we will mention in the next section about the API documents, Github helped identify and crosscheck differences between the actual API implementation and the descriptions provided in the team-generated interface documentation.

API Documents/Google Drive

Each of the team members has been using Google Drive for classes throughout the SWENG program, so it was a natural solution for sharing information that required documents or more coordination than a forum post. The team's use of Google Drive helped ensure success, allowing the collaboration of weekly document submissions to coordinating design integration. The team's Drive folder provided a centralized reference for system design documents, progress, and task assignments. The use of Drive documents to create an API reference document, was critical to the accomplishment of key integration goals that would have been much harder, if not impossible, to coordinate without a well-defined reference.

The team's efforts to document the API began as it was realized that it would be difficult to utilize a RESTful API without specific knowledge of the expected parameters, return format and range of messages, and possible error conditions that may be raised. As a result, from the first API functions that were implemented, we created a documentation system that described the purpose of the function, the method of implementation, parameter requirements and format, return message format, and possible return conditions. This allowed the team to decouple the design of the API from the implementation of the web and Android clients, decreasing the

amount of direct communication needed for team members to accomplish their respective taskings.

Eventually, it was realized that not all API functionality would be able to be designed by inspecting the system requirements document. Several API functions would depend upon constrictions made by the specific implementation of each client, the way the client would display and store data, and the information that was available to each client when the API function would be needed. This information was not readily available to the API team as they were designing functions, so the team created another document to track requests for API functionality. This document was not as formal and specific as the API documentation, but it provided a method for the client designers to request functionality they anticipated they would need and offer a reasonable interface of parameters that could be used to reach the goal. The request document allowed functionality requests to be made before they were needed, giving the members working on the API time to coordinate and clarify the needs of the function before development was delayed for lack of functionality integration details. It also provided a forum where each team member could discuss their needs and weigh in on the design decisions without impacting the finalized API documentation that would describe the way the function should be implemented. The API functionality request document also helped to limit extra work for the API development team. Early on, we quickly found out that it was hard for the API team to know what the end user team needed, so some time was spent on functions that the end user team wouldn't need. This document help to keep the API development team and end user team focus towards one goal.

Unexpected Challenges

Despite the planning and requirements development process, there were challenges the team did not anticipate and difficulties that couldn't be sufficiently planned for. As the team worked through the development of the system, the following hurdles provided particularly tough challenges to overcome.

The team has had discussions on how to manage the user request to join a team. If the team that the user is requesting to join is a "managed" team, the manager of the team needs to be informed of the request to perform the action (approval/disapproval) of that user. This can be tricky to handle with the client and android application, however, and the team had to come to a consensus that the manager will receive a notification when they login to either the app and view the specific team or web client and have the ability to perform the action from either vehicle at that point. The coordination of delivering the list of pending member applications and approving them using REST, where sessions are not kept explicitly, turned out to be more challenging than expected, requiring a series of coordinated data calls to the receiver once the client recognized the user as the manager of a team to check for pending users, followed by the manager action on these requests.

The process of deleting teams and team members has also provided an unexpected challenge. The deletion of a team must lead to the deletion of the corresponding event, member, tasks,

and so on in the database. The design of the foreign keys has made this cascading deletion more difficult to implement than expected. Designing tests to ensure that a team is not kept active after all the team members leave has increased the difficulties for managing the database logic. The system had to be designed to check for a number of conditions when a team member left a team, such as whether they are the last team member or the last manager of a managed team. In these cases, confirmation would be required before a team was deleted, requiring more coordination between the client applications.

Another challenge we faced was generally around the idea of a team. The two challenges above are an aspect of what made it difficult, but in general since the team concept was critical to our project, we had a hard time to reign in all the different features and functionality of the team module. For a week or two, it seemed we would endlessly come up with more features or complications the team module would need to worry about. Teams and each of their components required a number of interactions to be useful to the user, each of which required careful thought when implementing the operations. Even after we were ready to start testing different parts of the team, we still ran into issues during the integration tests. While we were testing, unexpected complications would continue to give us issues. Eventually, we would overcome this challenge and be able to complete the team module on both the API and end-user ends of the application.

The Android framework offers several UI “widgets” that can be built up to create interfaces for an application. Several features of the framework do not work well on certain combinations of widgets or app behaviors. The team discovered late into the design that several of their interfaces required significant redesign to allow the correct behaviors, occasionally requiring the implementation of custom widgets and modified views, and other times, selecting different widgets that were more compatible with the desired functionality. These challenges were all overcome, but it was an unexpected roadblock that slowed the progress on completing app functionality.

Working with mobile devices provided an interesting challenge. Even with the abstraction from hardware provided by the Android framework, the team found that different devices behaved slightly differently. This became an issue for capturing receipt images. The initial solution worked as expected on the phone Peter was using for development and debugging, but when other team members tested the camera functionality on their devices, we found various behaviors across the various devices, leading to unusual behaviors that were difficult to troubleshoot. Eventually, we learned that not all camera software returns images in the way described in the Android documentation, including the newer versions of the basic Android camera app! In addition to this, the process to create files and access file locations had been updated without much guidance from the official Android developers resources. This unexpected challenge was particularly acute since the camera use to add receipts was one of the last features to be implemented and ran very close to the final presentation timeline by the time the team successfully debugged the issue and were able to field a solution. The problems encountered receiving images from the camera threatened to force the team to drop this

functionality from the presented product, but we were able to find a solution and test it on all the devices we had available to us successfully in the days leading up to the presentation.

Lessons Learned

The project provided plenty of learning experiences for the team. The takeaways the team gained were multi-faceted and spanned a range of issues from project management to specific technology choices. To address the wide range of issues the team felt were important, this section is divided to focus on the primary areas where the most progress was made.

Requirements

The team was fortunate that the requirements were very stable throughout the progress of development. The team had a very good idea of how we wanted the system to work early on and spent a lot of time ironing out the logic that would determine the state of a team and team member relationship. During development, creating mock user stories that allowed us to mentally step through the anticipated use of the system was invaluable to the achievement. This greatly helped as we added functionality, because it was usually clear how it should operate and everyone was usually on the same page. The ambiguities that we did discover were addressed quickly by the team as misunderstandings were found and the team almost always came to a quick consensus on the desired behavior of the system.

Even with stable requirements, there were some functions that were not described sufficiently during the initial requirements and a few additional functions that the team found they wanted to add to improve the user experience. Additionally, as mentioned in the unexpected challenges section, even stable, well thought-out requirements had some ambiguities when it came to managed teams and team deletion that turned out to be more difficult to tackle than initially anticipated.

The takeaway on requirements that they team found was that conceptual walkthroughs of use scenarios were well worth the effort and time. The walkthroughs helped identify most challenges early and kept the team working toward a common functional goal that was usually abstracted from the specific interface or technologies involved.

Coordination and Integration

Coordinating tasks were a major challenge due to the scope of the project, level of integration required, and, most of all, because the team was geographically separated. Early in the project, this was mitigated by assigning team members to different parts of the system, but, as discussed, the API buildup required significant interaction to keep the functionalities aligned. Most of the project was conducted with only bi-weekly real-time meetings. Using the time after the assigned bi-weekly progress updates, the team was usually able to map out tasks and clarify requirements and desired functionality that there was misunderstanding and miscommunication about. On top of this, some of the more involved debugging efforts, particularly those resulting from API integration and network issues, were augmented by the use of text chats that took place near-real time as problem solutions were tried out and logs that

were not accessible to everyone were examined for the source of the flaws. This allowed such problems to be resolved much more rapidly than forum posts and email.

The team made great use of the tools available in Google Drive and also github. We found, however, that github had trouble merging the various branches when a lot of work had been done to one part of the program which the master had been updated for others. For this reason, the use of a repository was less helpful than hoped, but still allowed team members to keep up with the progress of other parts of the system by providing an online history and the ability to view files for any part of the system. This was augmented by the Task reports the team devised on Drive, which provided the easiest way to track progress.

As previously discussed, the two-way API request and documentation was instrumental in allowing the team to work together on closely integrated parts of the system without requiring scheduled meetings and work-schedule coordination, which proved difficult due to the different time zones and shifting schedules everyone on the team dealt with. The team effectively used the comment function of Drive documents to bring up issues and resolve them. The ability to receive email updates on Drive documents helped as well. Every document the team created in Drive ended up having extensive comment conversations that helped clarify the desired functionality of the system and identify parts of the system that were creating faults or not producing the correct behavior.

The team learned that two-way function requests and formal functionality description was essential to keeping the team coordinated when real-time meetings and updates were impractical. The effective use of comment conversations also helped, but occasionally led to further disputes or miscommunication, so using these, it is important to identify this and clarify in real-time if necessary.

Timeline Planning

The project ended up meeting the design timeline, but not without some progress hangups and surges of work. Planning the timeline was very difficult due to the impossibility of anticipating how long work would take on features the team had never worked with before, such as external APIs, portions of the Android framework, and AngularJS, schedule availability, work conflicts, and a second class during the initial portion of the project. The team kept up with almost all of the weekly task assignments, but these were not paced out for the entire project. Focusing on immediate tasks was helpful, but meant that the overall pace was difficult to gauge. This was demonstrated during the half-way task report where the team reported only 35% completion. Luckily, the conclusion of the second academic class opened up a lot of time to contribute to the project after stalling progress for the completion of final reports. The team did not anticipate the impact this class would have and, as a result, had to make up for lost time in the following weeks. This was achieved, but the overall progress planning was not really tracked in detail until the final four weeks.

The lesson here is that it would have been valuable to have a more thorough long-term schedule designed, but one that was flexible enough to adapt to unanticipated challenges. The team could have found tools that would have helped track this progress, allowing that such tools would allow for compression of some tasks to make up for delays early in the project. Such a tool would have helped significantly and should be incorporated into future project efforts. The two-week pace of progress reports did assist in short and mid-term goal setting, however, and helped the team stay on task and allowed the project to complete on time.

Advancing Our Knowledge

The team started out with background in most of the areas required for the project. There were sections that we knew from the beginning would require research and new techniques that we had not practiced before, however. In addition to the coordination and project management aspects, everyone on the team was able to improve their understanding of some technologies they worked directly with.

Database

The database used for the project was based on MySQL, which the team had worked with before. We discovered new tools that helped design and build the database more quickly. The database went through several iterations as the needs were clarified. David's work here began in a structure-centric mentality and eventually shifted toward supporting the specific queries from the server API. Initially with the structure-centric approach, the database was designed more like object-oriented software would be designed. Each type of data had its own table, and foreign keys would be used to represent connections between the data. Before we would move onto the heavy coding phase, we tried to think ahead as to what we really want to get out of each table from a system level and complexity perspective. This led to the first refactoring iteration to simplify the schema. We found that some of the keys initially chosen and table references made the more common queries difficult, even though the structure made sense in the relation of the objects involved. As a result, multiple restructurings had to occur, especially when new functionality was introduced, such as tracking user locations and recording expense reports, that forced the database to be redesigned. These restructurings helped to simplify a lot of functionality in both the API and end-user code.

Android Framework and Google APIs

Peter worked on the Android app development, and although he had experience designing simple project apps for Android previously, he had not had the opportunity to work with much of the functionality required for this project. The app targets the newest versions of Android, which include UI elements that Peter had never worked with extensively in the past. Learning the new parts of the framework was challenging, but provided the app with a much better user experience and was eventually mastered sufficiently to provide the desired UI interactions. Initially, some UI designs were avoided due to the requirement to build up adapters and custom interface elements, eventually, however, the app ended up implementing all of these aspects and utilizes a lot of customizations that extend or override Android standard UI components. This significantly improved Peter's understanding of the Android development concept and provides more powerful interfaces for the user. Peter had also never worked with networked

applications before, and had to learn how to make Get and Post calls using HTTPS, authentication, JSON objects, and even file transfer operations. Finally, working with Google APIs was a new field for Peter. The team was able to integrate the Google Maps and location APIs directly into the application, which provides necessary functionality for the app and a great learning experience that was fun to implement.

Server APIs and External APIs

Stephen led the API and server development. He did not have any experience with developing custom APIs or REST prior to this project, external API use was also limited. Experience with servers helped make the setup quick, but the API design required significant interaction with the client software. Through the project we had the opportunity to work with mobile applications, which have slightly unique operations, even over standard HTTP messages. Additionally, Stephen developed an authentication process for calls to the server, which provides protection against replay attacks and still conforms to REST. To accomplish authentication that was supported on all clients and the server required more investigation on javascript and java for the clients. Another addition was the use of external APIs through the implementation of calls to FourSquare. This experience added a lot of value along with the benefit of adding “real world” data to the application. Along with the internal and external APIs, Stephen engineered a framework to allow the server to act as a email relay to provide a medium for the Away Team server to email its users. These email features required extensive research on options for DNS, Linux, sendmail, and postfix email applications. David also contributed to the API buildup, giving an opportunity to practice PHP programming and integrating with a database.

David assisted with the APIs that both the android app and web client would use. While David had experiences with web development, he was relatively new to JSON, Get and Post call handling, and REST interface. At first, this inexperience was obvious in the code that was written had some issues and showing confusion between Get and Post. Learning these different concepts proved challenging and difficult at first. As the project advanced, David felt more comfortable working with REST, JSON, and Get and Post calls. Eventually, he would have his own local test environment to test the API functions he produced. By that time, David was able to write code that made more sense and felt comfortable enough to debug through the code and quickly identify issues. The knowledge gained early on working with REST, JSON, and Get and Post calls helped in the end of the project to quickly implement the final desired modules requested by the end-user development team.

Web Application

In the web application, Clay opted to try out a new language, AngularJS, a Javascript framework developed by Google. This was Clay’s first dedicated experience to developing a full scale web application from start to finish, although he had experience with maintaining web applications. His previous experience was largely dealing with more dated web technologies with his current employer. Clay chose to use AngularJS as it allows the flexibility and scalability of the MVC framework with the ease of use of Javascript. In Clay’s opinion, AngularJS is how Javascript

should have been written a long time ago. AngularJS was used in a combination with other frameworks such as AngularUI and Boostrap3 for styling and UI implementation. Yeoman, Bower, Grunt, and Node.js were all used in the development pieces in order to make the process as fluid and swift as possible. Yeoman and Bower were used to manage the project's dependencies for third-party libraries and Grunt was used in the build process in order to minimize the amount of Javascript code, import the correct dependencies, and run all the unit tests that were developed with the Karma framework. Clay worked several Google APIs into his web application, including integrated map displays and location collection. He was able to develop a responsive, cross-platform application that was capable of scaling to different devices with different resolutions such as mobile devices. The responsiveness allows those who may be on the go but don't have an Android device the ability to use a dedicated web client for their mobile device.

Conclusion

The team is very satisfied with our work on the Away Team project. We accomplished our goals and met the initial requirements despite several challenges. The project also provided plenty of opportunities to learn to work in a distributed team as well as improve our skills in particular technologies. We were able to try out new tools to help meet our challenges and feel that we succeeded in creating a challenging, integrated system that incorporates networked components that complete a complex task.

Measurement of Success: Goals Met

The team met all of the goals set at the beginning of the project. There were requirements that needed clarification and some functions that were modified to fit the limitations of the technologies, but none of this impacted the final functionality of the system. At times, the schedule threatened to prevent the complete development of all the desired functionality, but through significant efforts, the team was able to catch the schedule and deliver all the requirements on time, even providing a week to test and debug remaining issues. To highlight the accomplishment, we will step through each of the requirements placed at the beginning of the project:

Nonfunctional Requirements

- **The system shall be able to store information on at least 100 teams**
    - The system stores multiple teams and has no defined limit on the total number of teams. The storage of data is not a limiting factor to the system that we were able to determine.
- **A managed team shall have at least one manager**
    - The system is designed to delete a managed team when the last manager leaves after providing a warning to the manager about the result of the action and requiring confirmation. When a managed team is created, the creator is automatically assigned as the default manager.

- **A managed team shall require permission granted by a manager for each user who joins**
    - Managed teams provide a “Pending Users” list that prompts the manager to accept or reject membership applications. When a member is in a pending status, they see the status in their teams list and may not view the team information.
- **The system shall support the use of managed and unmanaged teams**
    - The system allows users to join or create managed or unmanaged teams.

Functional Requirements

- **The system shall utilize username and password to identify users**
    - All user authentication actions use username and a secret token to prevent replay attacks after the initial authentication using username and password. This requirement is met and the system provides enhanced security.
- **The system shall use username to reset a user’s password using email notification**
    - When a member indicates that they forgot their password, a unique, strong, random password is emailed to their stored email to enable secure login.
- **The system shall collect user location at intervals**
    - The system collects user location when it is made available through either the Android application or web client interfaces
- **The system shall allow users to act as members of one or more teams**
    - The system allows users to join more than one team and provides a list of teams the user belongs to so that they may easily switch between teams.
- **Users shall be able to create, join, and leave teams**
    - These options are always available to any user on Android and web clients.
- **Team managers shall be able to authorize users to join a team, remove users from a team, assign current team members as additional team managers, and leave the team**
    - The team manager privileged actions are provided through both interfaces, giving the manager rights to accept or reject membership applications, promote a member to manager status, and leave the team, which will result in team deletion if they are the last manager. The last manager leaving is warned about the result of their action and asked to confirm.

The project was definitely a success. It was completed on time and met all the requirements the team identified in the requirements document. The final system is well integrated and works on a wide range of devices and interfaces.