**CS673 Software Engineering**

**Team 2 - Communication Tool**

**Project Proposal and Planning**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Team Member** | **Role(s)** | **Signature** | **Date** |
| Laura Kocubinski | Team Leader | *Laura Kocubinski* | 10/3/19 |
| Hang Shi | Backup Team Leader  Requirement Leader | *Shihang* | 10/3/19 |
| Xi You | QA Leader | *Xi You* |  |
| Jhuanderson Macias | Design and Implementation Leader | *Jhuanderson Macias* | 10/3/19 |
| Mikhail (Misha) Chertushkin | Security Leader | *Mikhail Chertushkin* |  |
| Bofeng (Beven) Liu | Configuration Leader | *Bofeng Liu* | 10/3/19 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| 0.1 | Laura Kocubinski | 9/19/19 | First Draft |
| 0.2 | Laura Kocubinski | 10/3/19 | Second Draft |
|  |  |  |  |
|  |  |  |  |

[Overview (Misha)](#_gjdgxs)

[Related Work (Misha)](#_30j0zll)

[Proposed High Level Requirements (Hang)](#_1fob9te)

[Functional Requirements](#_3znysh7)

[Nonfunctional Requirements](#_2et92p0)

[Implemented Features](#_tyjcwt)

[Management Plan (Laura)](#_3dy6vkm)

[Process Model](#_1t3h5sf)

[Objectives and Priorities](#_4d34og8)

[Risk Management](#_2s8eyo1)

[Monitoring and Controlling Mechanism](#_17dp8vu)

[Schedule and Deadlines](#_3rdcrjn)

[Quality Assurance Plan (Xi)](#_tiz7dffsvet3)

[Metrics](#_q0ornanw12rl)

[Standards](#_le20cnnk3e11)

[Inspection/Review Process](#_1ksv4uv)

[Testing](#_2jxsxqh)

[Defect Management](#_3j2qqm3)

[Configuration Management Plan (Beven)](#_blu4p9rlgarq)

[Configuration Items and Tools](#_4i7ojhp)

[Change Management and Branch Management](#_2xcytpi)

[Code Commit Guidelines](#_1ci93xb)

[Integration and Deployment Plan](#_wczy24e89azn)

[References](#_3whwml4)

[Glossary](#_2bn6wsx)

# Overview (Misha

The purpose of this project is to build a web-based application that facilitates real-time communication between members of a group or team. Such a team could be organizational (i.e. a software development team) or academic (i.e. a group of medical students). Similar applications include Slack, FlowDock, and Telegram.

The main features of this application will be:

* Direct messaging. A user shall be able to send messages directly to another user, one-on-one.
* Low latency time. A user shall be able to communicate with another user in real-time with little or no latency.
* Guaranteed message delivery. When a user sends a message, the message shall be delivered.
* Simple sign-up and sign-in process. A new user shall be able to register for and sign into the application with ease.

The optional, nice-to-have features will be:

* Chat rooms. A user shall be able to create a room/spaces/chat for internal discussion among multiple users.
* Emojis, images and attachments. A user shall be able to send and receive emojis and images, and attachments.
* Translation. If users speaks a different language, the application shall translate the messages on-the-fly between all users into one common language (i.e. English).
* History management and message statistics. A user shall be able to see their entire message history and observe who they interact with frequently.

# Related Work (Misha)

Today, the best communication applications include WhatsUp, WeChat, Telegram, Skype/Skype-For-Business, Slack, FlowDock, and Viber.

However, unlike some of these examples, this project plans to develop a *web-based* application (not mobile or desktop). As such, the most relevant example application is the web-based version of Slack.

In recent years, Slack has gained popularity rapidly; just this year, Slack went public. Slack allows team members to collaborate via direct or group messages. Additionally, Slack supports a lot of useful features. Such features include file sharing, fast message delivery, robust management of workspaces/channels, emoticons, and the ability to upload custom emoticons.

Another pertinent application is the web-based version of Telegram. Telegram has the best cryptography engine among its competitors and is used by a diverse set of people across the world. This project does not plan to implement a cryptography engine similar to Telegram, but security and cryptography issues will be considered carefully.

# Proposed High Level Requirements (Hang)

## Functional Requirements

* + 1. Essential Features - the core features that you definitely need to finish
       1. A user shall be able to send and receive messages directly with another user, one-on-one.
       2. A user shall be able to register and log in with ease.
       3. A user shall be able to report a problem (bug) with the website.
    2. Desirable Features - nice features that you really want to have too)
       1. A user shall be able to send and receive messages in a group chat.
       2. A user shall be able to login using different functions, such e-mail and phone number.
       3. A user shall be able to send and receive emojis.
       4. The application shall block a user from logging in if they have 10 failed attempts. A user shall wait 5 minutes before another attempt.
       5. When registering, a user shall enter the desired password twice for confirmation.
       6. A user shall login with email and password.
       7. A user password shall meet the following criteria
          1. The password shall not be too similar to the email/username
          2. The password shall contain at least 8 characters
          3. The password shall not be a commonly used password
          4. The password shall not be entirely numeric
       8. The application shall allow a user to reset their password via email.
    3. Optional Features - additional cool features that you want to have if there is time
       1. A user shall be able to send and receive images.
       2. The application shall suggest a user to re-join a chat they frequently participate in.
       3. A user shall be able to send and receive file attachments.
       4. If users speaks a different language, the application shall translate the messages on-the-fly between all users into one common language (i.e. English).
    4. Existing Features - not applied to a brand new project
       1. Tinder: the app which focus on 1:1 chat but not allows group emoji chat.
       2. iMessage: the iOS app which based on phone number but not allows interesting group invited.
       3. Venmo: the app focus on the payment but using the emoji as their attraction.

## Nonfunctional Requirements

* + 1. Quality
       1. Reliability
          1. The application shall allow at least 500 simultaneous users.
          2. The application shall allow message transfer with less than 10s latency.
          3. When an error is encountered, the application shall protect the user’s data.
       2. Performance
          1. A user shall feel satisfied that their messages and sent and received quickly; satisfaction is four stars or greater out of five.
          2. The tool shall allow data transfer at least 78 percent influenced by the user’s internet quality.
          3. A user shall be able to communicate with another user in real-time with little or no latency.
       3. Security
          1. The system should have basic protection from DoS attack.
       4. Maintainability
          1. The application shall collect user feedback and provide this feedback to the development team for an enhanced user experience.
       5. Portability
          1. A user shall use the application in a web browser executed on mobile and desktop. The application shall be able to be ported to mobile or desktop.
          2. A cloud database service shall allow easy transfer of user data to a new database service.
          3. The application shall exist in a mobile version.
    2. Constraints
       1. Tools and Languages
          1. The application shall allow communication in English.
          2. The front-end and back-end will be implemented with Python.
    3. External Interface
       1. The application shall provide a link share to Instagram, Facebook, iMessage, Twitter or other relevant social platforms.
    4. User Interface
       1. The tool shall be user-friendly; user rating must maintain above 4 out of 5 stars.
       2. The organization web page shall provide GitHub open source address for developers.
    5. Error Handling

1. The application shall use a core function named JIT (Just in Time) for developing processes and fixing errors.
2. The released application shall not have a development or logic problem.
3. The development team shall also provide ‘quick response’ (within 24 hours) to a reported error.

## Implemented Features

* + 1. Iteration 1
       1. iOS app
          1. iOS platform user register (including name + password + e-mail address)
          2. iOS platform user login (including name + password)
          3. C-S architecture technology using
          4. Bootstrap web front page choosing
       2. Web app
          1. User registration (username and password)
          2. User login (username and password)
          3. User can join a chat room by name and send messages to other users in that chat room
          4. Django-AWS server side test
          5. Web application platform structure diagram

# Management Plan (Laura)

## Process Model

* + 1. For this project we will use an agile-like iterative and incremental development process.

## Objectives and Priorities

* + 1. The overall objective is to deliver a high-quality software tool by 12/5 that facilitates real-time communication between team members.
       1. Each iteration (i.e. project milestone) and associated documentation should be completed on time.
       2. Each iteration should deliver high-quality software via testing and timely defect resolution.
    2. By the end of the project, all core customer requirements should be implemented.

## Risk Management

* + 1. Risks will be continuously assessed throughout the project development process. When a risk is identified, it is important to qualify and quantify said risk. A risk can be categorized as follows:

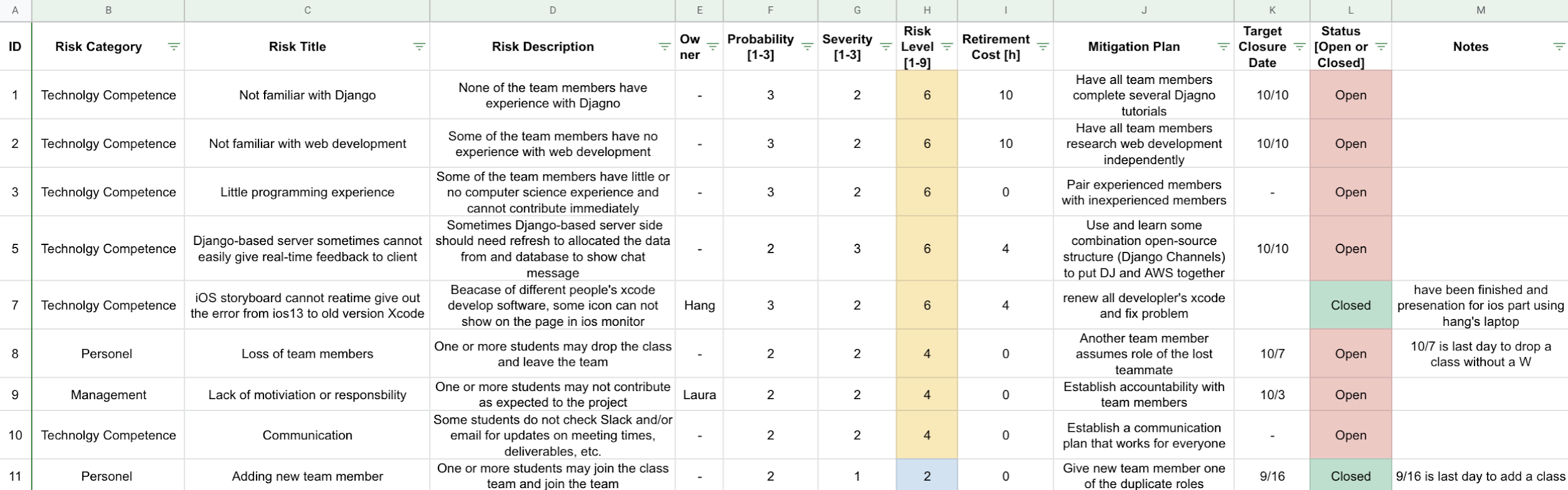
|  |
| --- |
| **Risk Categories** |
| Management |
| Implementation |
| Testing |
| Integration |
| Deployment Rush |
| Specific Technical Issues |
| Improper Scope |
| Requirements |
| Communication |
| Personel |
| Technology Competence |

To quantify the risk, a probability and severity level (1 - 3) is assigned.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Risk Rating Criteria** | | | | | |
| **Probability** | | | **Severity** | | |
| **Rating** | **Likelihood** | **Description** | **Rating** | **Impact** | **Description** |
| 1 | Rare or Unlikely | Could occur at some point | 1 | Minor | Minor impact to project |
| 2 | Possible | Might occur at some point | 2 | Moderate | Some impact to project |
| 3 | Likely or Very Likely | Will likely occur at some point | 3 | Major | Major impact to project |

Together, these values are multiplied to generate an overall risk level, which aids to identify the highest level risks.

A [workbook](https://drive.google.com/open?id=1GTWBbLVAPDI4BMpjY49GN0F6Y_pPP_Lx) is maintained to track the project risk status.



## Monitoring and Controlling Mechanism

* + 1. Several tools will be used to monitor and control the project.
       1. GitHub will be used for revision control and defect tracking.
       2. Slack will be used for inter-team communication and alignment. Each team member is expected to check Slack daily.
       3. Google Docs will to share project documents:
          1. Meeting minutes (action items)
          2. Weekly progress report
          3. Risk management workbook
          4. SPPP
          5. SDD
          6. STD
          7. Project schedule
       4. Pivotal Tracker will be used to manage software requirements (i.e. user stories) and to assign said user stories and tasks to team members.
    2. Additionally, the entire group will meet at least once a week in person after class on Thursday in PSY B33. If a second meeting is required, the group will meet remotely on Sunday at 6 PM using Live Classroom (Zoom). Additional meetings for smaller groups will be scheduled on-the-fly.

## Schedule and Deadlines

* + 1. This project consists of an initial planning phase (iteration 0) followed by five iterations or “sprints”. Each iteration will conclude with a working, quality software prototype. Each iteration can be thought of as a “mini project,” as it will require planning, requirement analysis, design, coding, and testing. At the end of each iteration, a short presentation and demo is required.
    2. The [high-level deadlines and project schedule](https://drive.google.com/open?id=1T4Pb7-Xr4q3azc5myYx4F7_eK_pcMgazXQsvus5BFGs) diagram may be found below.

|  |  |  |
| --- | --- | --- |
| **Week** | **Start Date** | **Project Deadlines** |
| 1 | 9/5 | Iteration 0 Starts |
| 2 | 9/12 |  |
| 3 | 9/12 | Iteration 0 PresentationIteration 1 Starts  First Draft of SPPP Due |
| 4 | 9/26 |  |
| 5 | 10/3 | Iteration 0 Presentation  Iteration 1 Starts |
| 6 | 10/10 |  |
| 7 | 10/17 | Iteration 2 Presentation  Iteration 3 Starts |
| 8 | 10/24 |  |
| 9 | 10/31 | Iteration 3 Presentation  Iteration 4 Starts |
| 10 | 11/7 | Iteration 4 Presentation  Iteration 5 Starts |
| 11 | 11/15 |  |
| 12 | 11/21 |  |
| Holiday | 11/28 |  |
| 13 | 12/5 | Final Presentation  Iteration 5 End |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Schedule for Communication Tool | | | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | September | | | | October | | | | | November | | | | December | | | |
| Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  | 13 |  |  |  |
| Start Date | 9/5 | 9/12 | 9/19 | 9/26 | 10/3 | 10/10 | 10/17 | 10/24 | 10/31 | 11/7 | 11/14 | 11/21 | 11/28 | 12/5 | 12/12 | 12/19 | 12/26 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Major Milestones |  |  |  |  | P0 |  | P1 |  | P2 | P3 |  |  |  | Release | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Iteration 0 | Planning | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Iteration 1 |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Iteration 2 |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |
| Iteration 3 |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |
| Iteration 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Iteration 5 |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* + 1. A more detailed schedule for each iteration is proposed below.

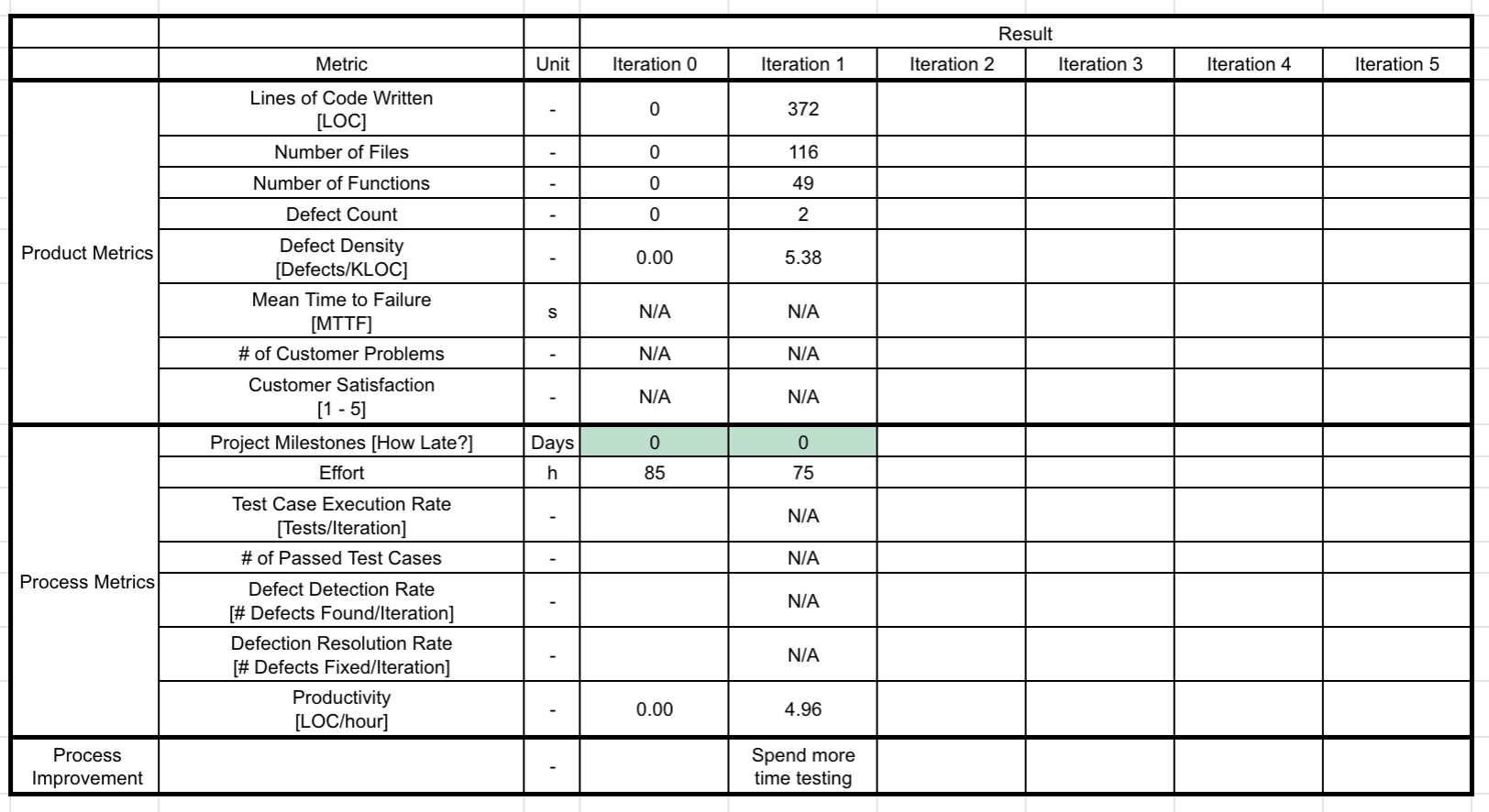
|  |  |
| --- | --- |
| **Day** | **Activity** |
| Day 1 | Prioritize User Stories for Iteration  Assign Tasks |
| Day 2 | Design & Implementation |
| Day 3 |
| Day 4 |
| Day 5 |
| Day 6 |
| Day 7 |
| Day 8 |
| Day 9 | Unit Test, Documentation |
| Day 10 |
| Day 11 |
| Day 12 | Pull Request: Peer Review Code, Final Test |
| Day 13 |
| Day 14 | Merge Code and Documentation into Master |

* + 1. Deliverables due at the end of each iteration are as follows.
       1. Presentation with demo of working software
       2. Requirements analysis with Pivotal Tracker
       3. Updated documents (SPPP, SDD, SDT, weekly report, meeting minutes)
       4. Source code (production and test code)
       5. Self and peer reviews

# Quality Assurance Plan (Xi)

## Metrics

* + 1. Product and process metrics will be compiled in a separate [Product and Process Metrics](https://drive.google.com/open?id=1KxHKXy2zatvqMQcWubNz62lWp78NjsRHPwLuyVd0Xqo) workbook at the end of each iteration. This workbook pulls information from other workbooks (such as the [Progress Report](https://drive.google.com/open?id=1s3vKeeQWJzEQ3uE9DJMFMs4nFmVRe6twNIb9Xs8iqz8) workbook), GitHub and shell scripts to pull product information.
    2. Product metrics
       1. Size and complexity
          1. Lines of Code (LOC)
          2. Number of files
          3. Number of classes
       2. Defect Count
       3. Defect density - the number of defects relative to the software size (i.e. defects/KLOC)
       4. Mean time to failure (MTTF) - amount of time elapsed between crashes
       5. Customer problems - total number of problems encountered by customers (in this case, the professor) while using the tool
       6. Customer satisfaction - how satisfied the customer is with the tool
    3. Process metrics
       1. Project milestones - number of days between a milestone’s schedule and the day on which it was actually achieved; adherence to schedule
       2. Effort - the number of man hours (or months) of development
       3. Lines coded per Man-Month (MMDefe
       4. Testing progress
          1. Test case execution rate
          2. Number of passed test cases
       5. Defect detection rate (per phase) - rate at which defects are found
       6. Defect resolution/repair rate - rate at which defects are resolved
       7. Productivity
    4. Results (to be completed at the end of each iteration)
       1. Curre nt results with notes for process improvement



## Standards

* + 1. Documentation
       1. The project documentation (SPPP, STD, SDD) should follow the templates provided.
          1. <https://drive.google.com/drive/folders/1y9cBPVK1S6sxGH6fMLFXdF9V9f6I9r3L>
       2. The Software Test Document (STD) may follow IEEE 829-1998: Standard for Software Test Documentation.
       3. The Software Design Document may follow IEEE 1016-1998.
    2. Code
       1. Python
          1. The standards for Python can be found at the following links; PEP 8 will be followed. Python version 3.7.4 will be used.

[https://www.python.org](https://www.python.org.the)

<https://www.python.org/dev/peps/pep-0008/>

For example, according to PEP 8, function and variable names should be all lowercase with words separated by underscores; mixed case is allowed to match a prevailing style

* + - 1. Database
         1. The standard for database will depend on the database used. This is not yet established.

## Inspection/Review Process

* + 1. Inspections (and also reviews) are verification activities and involve peer examination of project artifacts (requirements, design, code, etc.) to discover defects. The goal is to identify as many defects as early as possible. The following high-level reviews should occur **at least** once and be repeated as necessary. The purpose of these reviews are to uncover defects.
       1. Software requirement review (requirement leader)
       2. Architecture design review (design leader)
       3. Detailed design review (design leader)
       4. Test plan review (QA leader)
       5. Verification and validation review (QA leader)
    2. Code review
       1. All code will be peer inspected (by at least one other person) before the end of each iteration. This will be facilitated by a pull request on GitHub. A pull request will require at least one other team member to review the set of changes and discuss potential modifications before the changes are merged into the master branch.
       2. This review should cover:
          1. Variables

Do variables have meaningful names?

Are hard-coded numbers used instead of constants?

Are all variables used?

* + - * 1. Functions

Do functions have meaningful names?

Are all parameters used?

* + - * 1. Correctness (Syntax)

Are parentheses matched?

Are brackets matched?

* + - * 1. Initialization

Are variables initialized before use?

* + - * 1. Loops

Do all loops terminate?

Do break and continue work properly?

* + - * 1. Dynamic Allocation

Is every dynamically allocated piece of memory deallocated?

* + - * 1. Pointers
        2. Comments

Is the code properly commented?

* + - * 1. Defensive Programming

Are checks made for errors (i.e. divide by zero, illegal data)?

* + 1. For the sake of time, a formal inspection (planning, preparation, meeting, rework, follow-up, improve process) will likely not be followed. However, all inspections will be guided by these principles:
       1. Peer process
          1. Conducted by a group familiar with the artifact under review
       2. Specific roles
       3. Defect detection instead of defect repair
          1. Inspections should focus on identifying defects, and specifically exclude any discussion of their repair
       4. Use of checklist
       5. Artifact readiness
          1. The artifact under review should be in the best state of readiness
       6. Adequate preparation
       7. Metrics collection
          1. Metrics are collected and analyzed
       8. Time limit

## Testing

* + 1. A document will be used to summarize and link to all test results.
    2. Some of the different tests can be found below. Testing will be performed during every iteration. Testing will first be executed at the unit level. As units and integrated and modified, regression and integration testing will be performed. As the software reaches maturity, functional, nonfunctional, and acceptance testing can be performed.
       1. Unit testing
          1. Individual units of code should be tested *before* assembly.
       2. Regression testing
          1. If there is a new part added into original part (e.g. new functionality), repeat testing of the original and all its dependencies should be executed.
       3. Integration testing
          1. When assembling modules, integration testing should be executed to ensure the parts work together correctly.
       4. Functional testing
          1. Tests how customers will use the product in real life.
       5. Nonfunctional testing
          1. Nonfunctional testing addresses performance, load/stress, usability, recoverability, compatibility, security, installation, serviceability.
       6. Acceptance testing
          1. This tests that the customer requirements are satisfied.

## Defect Management

* + 1. Defects will be classified with severity, priority and type:
       1. Severity per IEEE standard 1044.1-1995
          1. Urgent - Failure causes system to crash, unrecoverable data or jeopardizes personnel
          2. High - Causes impairment of critical system functions, and n workaround solution exists
          3. Medium - Causes impairment of critical system functions, though a workaround solution exists
          4. Low - Causes inconvenience or annoyance
          5. None - None of the above
       2. Priority - the order in which the team plans to address it
       3. Type
          1. Omission
          2. Unnecessary
          3. Non-conformance with standards
          4. Inconsistency
          5. Unclassified
    2. To manage defects we will use the GitHub issue tracker.
    3. According to severity, priority, owner(s), and target completion date will be assigned. The issue tracker should be reviewed at least once a week with the entire team.
    4. To track the bugs in a user story we will use “Pivotal Tracker” and discuss how to improve the tool.

# Configuration Management Plan (Beven)

## Configuration Items and Tools

* + 1. Version control
       1. GitHub
    2. Defect management
       1. GitHub

## Change Management and Branch Management

* + 1. Change Management
       1. Identify which part(s) require a change and how the parts must be changed.
       2. Communicate the ideas for the changes with team members.
       3. Obtain approval from team leaders before making changes.
    2. Branch Management
       1. The GitHub flows consists of several steps: creating a branch, adding commits, opening a pull request, discussing and reviewing your code, testing and merging.
       2. Master branch is the definitive branch.
       3. We have other branches, such as a development branch, to experiment and develop code. The process of making commits, which means add, edit and delete a file in a feature branch, is recorded for all to see. Each commit should have an associated message that help team members understand what you have done and why.
       4. We use a pull request to start discussion and a code review the changes in the commit. Meanwhile, specific team members or leaders should give feedback.
       5. Next we, we deploy a branch for final testing. If this branch causes issues, just roll it back. If the commits have been verified, the new code could be merged into the master branch.
       6. Before merging other branches into the master branch, the team leaders must approve of the pull request.

## Code Commit Guidelines

* + 1. All commits must be commented clearly and meaningfully. Information in commit should explain:
       1. Error(s) fixed
       2. Why a change was made (i.e. re-factoring)
       3. Features/functionality added
       4. Limitations of current code

## Integration and Deployment Plan

* + 1. Individual units should be tested before integration.
    2. Test the units as a whole.

# Reference

* 1. Braude, E. J., & Bernstein, M. E. (2016). Software Engineering: Modern Approaches. Long Grove: Waveland Press.

# Glossary