Table 1: Revision History

Date	Developer(s)	Change
2022-02-03	Akram Hannoufa	Initial addition of Meeting Plan, POC plan, Tech, and Style
2022-02-03	Nathaniel Hu	Added in initial draft of the introductory blurb and Communication plan, Member Roles and Git Workflow plan sections
2022-02-04	Ganghoon Park	Added in location of project Gantt Chart to Project Schedule section
2022-02-04	Nathaniel Hu	Made a few corrective edits
...	...	...

SE 3XA3: Development Plan
Flag Generator

Team #2, Team Jakriel
Akram Hannoufa, hannoufa
Nathaniel Hu, hun4
Ganghoon Park, parkg10

The purpose of this Development Plan document is to outline all of the information necessary to develop the project. This project document details the specifics of the team meetings, team communication, and other team dynamics. In addition to that, it outlines the Git Workflow plan, the Proof of Concept Demonstration plan, and other aspects related to the implementation and technology required. Finally, it outlines the tentative project schedule and the project review (to be filled in at a later time).

1 Team Meeting Plan

- **Meeting Day:** Wednesday
- **Frequency:** Weekly
- **Time:** 5pm-6:30pm (1.5hrs)
- **Location:** Microsoft Teams Call (maybe switch to in person library meeting later)
- **Roles:**
  - **Scribe:** Nathaniel Hu
  - **Agenda Consultant:** Akram Hannoufa

2 Team Communication Plan

The following channels of communication will be used in the following instances where it is appropriate to communicate the specified information:

- *Messenger Group Chat*
  - low to mid-priority questions

  - declaration of intentions to edit a file
  - other low to mid-priority communication
  - late arrivals to labs or meetings

- *Email*
  - transfer of information not pushed to the repository
  - clarification of preferred merge conflict resolution
  - other low-priority, non-urgent communication

- *Teams Group Chat/Meeting*
  - meeting discussions and questions
  - urgent messages

- *Text/Phone*
  - urgent questions, emergencies
  - no reply to other communication channels (i.e. email and teams group chat/meeting)

3 Team Member Roles

- *Akram Hannoufa*
  - developer
  - testing expert

- *Ganghoon Park*
  - developer
  - documentation expert

- *Nathaniel Hu*
  - developer
  - git expert

4 Git Workflow Plan

4.1 Branch Types

The Git repository will be composed of the following branches:

- A **main** branch, which will contain the version of the code in production

- one **personal** branch per team member, which will be named the team member's MacID; it is on these branches where most of the development will occur
- one **shared** branch for main development called "develop"; it is on this branch where all work from the personal branches are integrated together, tested and verified before being merged into the main branch
- one **shared** branch per documentation deliverable, which will be the name of the documentation deliverable in lower case followed by the word "work", with underscores in place of spaces (e.g. "dev_plan_work"); it is on these branches where documentation development will occur, mainly to proactively prevent merge conflicts

4.2 Pushing and Pulling

- **documentation deliverable** branches will be synchronized with the **main** branch, but will only be merged into it when the documentation deliverable being developed is ready for a release
- **personal** branches will be synchronized with the **develop** branch, and will only be merged into the **develop** branch when they don't contain non-functioning/faulty code (to the developer's best knowledge)
- the **develop** branch will be synchronized to the **main** branch, and will only be merged into the **main** branch when it doesn't contain non-functioning/faulty code (to the developers' best knowledge)
- merging ready code into the **develop** or **main** branches (henceforth called the **target** branch) will work as follows:
  1. push the work on your **current (personal)** branch to its corresponding remote repository to have a back-up of your current code in case there are problems when merging
  2. switch over to/checkout the **target** branch, pull from the remote repository to ensure your local repository is up to date; check to ensure there are no merge conflicts, and commit any changes as needed
     - if there are any merge conflicts, notify the relevant team members and resolve them before continuing
     - coordinate with team members to avoid having them push other changes to the **target** branch as you are merging ready code/resolving merge conflicts
  3. merge the work from your **personal** branch onto the **target** branch, and resolve any merge conflicts
  4. commit and push your merged work to the **target** branch's remote repository

    - it is good to notify all developers using the **target** branch (or all developers if the **target** branch is the **main** branch) that you are pushing

- merging ready documentation deliverables work into the **main** branch will work as follows:

    1. do the **documentation deliverables** work on a file separate from the shared documentation deliverable file (unless you will be the first person to merge your work to the **documentation deliverable** branch)
    2. coordinate with other team members to ensure that only one member will be pushing their changes to the remote repository
    3. pull from the **documentation deliverables work** branch's remote repository, and ensure your local copy matches the one on the remote repository
    4. add your changes into the **shared** documentation deliverables file and commit those changes
    5. check with other team members and get the okay to push changes to the remote repository
    6. once the okay has been given, push your work to the remote repository, and ensure all team members can see the changes made and have pulled the changes to their local repositories **before** they add their changes to the same file
    7. repeat steps 2-6 until all changes to the **documentation deliverables** file have been made by all team members
    8. check over the **documentation deliverables** file, and merge the **documentation deliverables work** branch into the **main** branch once the team okay has been given
        - the documentation deliverables work will bypass the **develop** branch when being merged into the **main** branch
        - the **develop** branch will be synchronized with the **main** branch, and will thus be regularly updated with the documentation deliverables work that has been completed

4.3 Tags and Milestones

- tags will be used to mark milestones in the project's code development
- the initial version tags will be "v1.0"
- milestones will be determined based on the Gantt Chart
- additional milestones may be created throughout development, ideally with unanimous team agreement

5 Proof of Concept Demonstration Plan

5.1 Project Size

The original repository for pagan contains around 900 lines of code, which originally was a concern of being too small of a project to pursue over the term. However, by greatly expanding the generation possibilities (ie. many different flag options), and by building a grpahical user interface for generating the image, the scope of the project will be large enough for the term. Beyond implementing these features, code documentation, organization, and modularity will also be enhanced.

5.2 Testing

Given the graphical nature of this project (generating image files from an input string), testing whether or not 2 images are the same or not will likely require the use of external libraries. Some research will be into seeing if image comparison can be done using libraries such as openCV and numpy, thus allowing for automated testing of image equality. If automated testing for image similarity is not feasible, then a manual/visual inspection method of testing will need to be used. Additionally, checking the quality of the generated images, and testing of the GUI will likely need to be done through manual inspection. However, all other components of the project (ie. hashing, string input, etc.) will be tested in an automated fashion.

6 Technology

6.1 Programming Language

Python 3 will be the language used for the project as the original source project is written in Python, and it is a language all members of the team are familiar with. Additionally, Python has many libraries (hashing, graphics, etc.) that will be useful/nice-to-have as well as necessary to complete the project.

6.2 IDE

The group will use PyCharm as the main IDE for the project. This is an IDE that was developed specifically for use with the Python language and has built-in features that will support and aid in the development process (ie. detailed UI). However, if a group member is more familiar and productive with another IDE, then that IDE will be used by that member.

6.3 Testing Framework

This project will use the Python unittest library for unit testing. This framework allows for test automation, efficient setup/cleanup code and modularized testing.

Additionally, group members have worked with the unittest framework before. unittest documentation: https://docs.python.org/3/library/unittest.html

6.4 Program for Document Generation

The tool used for document generation will be Doxygen, which will generate the required code documents using the annotations (comments) included in the code. Doxygen documentation: https://www.doxygen.nl/index.html

7 Coding Style

A PEP-8 coding style will be followed for this project. All naming and commenting conventions will be adhered to, as well as other guidelines set out by the PEP-8 standard. PEP-8 documentation: https://www.python.org/dev/peps/pep-0008/

8 Project Schedule

The Project Schedule Gantt chart can be found at the following location:
../ProjectSchedule/3XA3_L01_GRP02_Gantt_Chart.pdf

9 Project Review