Project Imacs: Development Process Document

Group 16:
Kelvin Lin
Jiahong Dong
Liam Casola
Varun Hooda
Mikolaj Hrycko
Danish Khan
Prince Sandu
Baltej Toor

November 30th 2018

Contents

1 Process Workflow

The following documents Team 7's workflow processes. Since Project Imacs is a fully software-based project, all of the subsystems will be designed, implemented, and tested in a similar fashion outlined in this section. Project Imacs is split into 4 core features: the file management system, handwriting recognition and general inputs, data visualization, and user experience. Team members are split such that each pair of student "owns" one of the 4 core features, or a key aspect of overall software architecture design. Ownership is defined as the responsibility for ensuring that an appropriate amount of work is put into completing the feature by the overall team. All team members are expected to contribute to all 4 features, as expalined in Roles and Responsibilities.

1.1 Overview

All progress made in Team 7 begins with communciation between team members through a weekly meeting or through the online messaging platform Gitter. Once issues or opportunities have been identified, a issue ticket is created on Github's issue tracking software, and it is assigned to a team member based on pre-assigned responsibilities to oversee its completion. The assigned team member will be responsible for completing the assigned task, or decomposing it into smaller tasks and assiging it to other members. In completing the issue, the team member may use a variety of tools including, but not limited to, textbooks references for designing the software, a programming language (e.g. HTML, CSS, JavaScript, TypeScript) and an Integrated Development Environment for software development, and a web browser as well as automated testing tools for software testing. Once the assigned task is completed, it is reviewed by 2 members of the team, randomly selected, who did not participate in resolving the issue ticket. The team member assigned to the issue ticket is responsible for making any suggested changes by the 2 reviewers. Once the reviewers are satisified with the changes, then the work is merged into the master branch where all of the final source code and documentation is stored.

1.2 Tools Used

The tools that Team 7 expects to use are listed below. As the project evolves, additional tools may be introduced. All tools used shall be the latest publically available stable version, unless otherwise noted.

1.2.1 Communication and File Sharing

Communication and file sharing tools are used to facilitate asynchronous communication among team members, and to allow for the remote transfer of files. The following tools will be used:

- Gitter: Used for team-wide communication between team members.

- Email: Used for formal communication between Team 7, the professor, and the TAs.
- Google Drive: Used for sharing drafts of documentation before they are formalized. Google Drive also allows for collaborative real-time work.
- Google Hangouts: Used for video conferencing in the event of bad weather or if an emergency meeting is needed.

1.2.2 Text Editors

Text editors are used to develop documentation for Project Imacs. The following text editors will be used:

- LaTeX: Used for writing formal documentation, as defined in Writing Formal Documentation.
- Google Docs: Used for writing informal documentation, as defined in Writing Informal Documentation.
- Microsoft Word: Used for writing informal documentation.
- Markdown: Used for writing informal documentation.

1.2.3 Integrated Development Environments

Integrated Development Environments (IDEs) facilitate the development of software. The following IDEs will be used:

- Visual Studio Code: Used for intensive coding tasks as it contains advanced features to assist with software development.
- Vim: Used for coding when the changes are minor or if the task needs to be completed quickly on a resource limited machine.

1.2.4 Issue Tracking

Issue tracking software is used to track tasks that need to be completed, issues identified, or changes that need to be made. The following software will be used for issue tracking:

- Github Issue Tracking: Used to keep track of tasks that need to be completed, and for assiging team members to be responsible for certain tasks.

1.2.5 Programming, Markup, and Scripting Languages

Programming, markup, and scripting languages are used to develop software for Project Imacs. The following languages will be used:

- HTML: Used for creating webpages and user interfaces.
- CSS: Used to style user interfaces.
- JavaScript: Used for programming features for Project Imacs.
- TypeScript: Used for programming features for Project Imacs.

1.2.6 Package Managers

Package managers allow external software packages and libraries to be integrated into the software without having to manually install each package. Package managers can also ensure that each team member will download the appropriatly versioned required packages for the project. The following package manager will be used:

- Node Package Manager: Used to automatically organize and install the appropriate JavaScript/NodeJS libraries for each team member.

1.2.7 Version Control

Version control will be used to organize team workflow and track all changes made to Project Imacs. The following version control software will be used:

- Git: Used for tracking all updates and changes made to Project Imacs. More information is available in the Version Control section.

1.3 Identifying an Issue, Change, or Opportunity

As the project progresses, issues and opportunities will become apparent and team members will be expected to address them. To keep track of all issues and opportunities, Github's issue tracking software will be used. Github's issue tracking software allows for issue tickets ("issues") to be created and assigned to team members who will be responsible for overseeing its completion. The steps to identifying an issue or opportunity is as follows:

1. Discuss the potential issue or opportunity with the team either during the weekly meeting described in Holding a Meeting, or through freeform communication through the team's online messaging platform, Gitter.
2. Create an issue on Github's issue tracking software with complete details describing the issue or opportunity, and the deliverables required to mark the issue as complete.

3. Assign the appropriate team member(s) to oversee the completion of the task, according to the Roles and Responsibilities section.

By the end of these steps, a meaningful issue should be created on Github's issue tracking software, and the appropriate team members should be aware of the issue.

1.4 Holding a Meeting

Meetings are to be held during the first SFWRENG 4G06 time slot without a scheduled meeting with the professor or the TAs. Meetings will be held in an ITB capstone workroom, and the BSB lecture room will be used if there is no space available in ITB. All members are expected to be present, prepared, and to meaningfully contribute during the meetings. The meetings will be held according to the following steps:

1. Everyone sits in a circle facing towards the center such that everyone can see and hear each other.
2. Kelvin will update everyone of the upcoming deliverables required for the course, and team members will be assigned tasks to complete the required deliverable.
3. Varun will update everyone of the technical tasks related to the implementation of Project Imacs, and team members will be assigned tasks to help further its development.
4. Going around the circle, each team member will talk about what they worked on, what they need help with, and what they will do the following week. If a team member encounters difficulties that can be resolved by another team member, then the issue will be jointly assigned to the two team members.
5. All team members will be asked if they would like to add anything or ask any questions before the meeting ends.
6. The meeting is adjourned.

At the end of every meeting, all team members are expected to know what the required deliverables are for that week, and what tasks they are expected to complete.

1.5 Writing Formal Documentation

Documentation will need to be written to facilitate communciation, complete course deliverables, and enable knowledge transfer within the team. Formal documentatation is defined as any document that satisfies one or more of the following conditions: consistent formatting is important for continuity with a

previous document, there is a high likelihood that the document will need to be compatiable with a future document, the docuement was previously an informal document that became formal later on, the document is a course deliverable and it is longer than 1 page, or the document is a course deliverable worth more than 1% of the final grade. This definition implies that a previously informal document may become formal later on, and when it becomes formal, it must be properly converted.

The following steps outline the procedure for writing formal documentation:

1. Ensure a full understanding of the task-at-hand before starting. Ask any questions needed to better understand what is required.
2. Create a document on Google Drive and create an outline for the deliverable.
3. Fill in the outline.
4. Send a message on Gitter and ask other team members to contribute by adding additional content and editing existing content.
5. Pull from the `docs` Git repository.
6. Create a folder for the new document following the Git Standards and Conventions.
7. Create a new LaTeX file in the folder.
8. Properly convert the Google Drive document into LaTeX, ensuring that the formatting is appropriate for the document, and is reasonably consistent with any previous documents.
9. Push the change into the Git repository.
10. Send a message on Gitter and ask other team members to contribute by adding additional content and editing existing content.

1.6 Writing Informal Documentation

Informal documentation is defined as any document that is not formal. Informal documentation will primarily be used for internal purposes, and will have little impact, if any, on the course grading. While the quality of the content should be on par with the quality of a formal document, the difference between a formal document and an informal document is that an informal document does not need to have consistent styling with other documents, so that the informal document can be produced in a faster manner.

The following steps outline the procedure for writing informal documentation:

1. Ensure a full understanding of the task-at-hand before starting. Ask any questions needed to better understand what is required.

2. Open a text editor (e.g. Google Docs, Microsoft Word, Markdown).
3. Write meaningful content into the text file.
4. Upload it to Google Drive.
5. Send a message on Gitter and ask other team members to contribute by adding additional content and editing existing content.

Additionally, if the informal document is a course deliverable that is 1 page or shorter, and it is worth less than 1% of the final course grade, then the following steps need to be taken:

1. Ensure that a reasonably sized serif font is used in the document.
2. Pull from the `docs` Git repository.
3. Create a folder for the new document following the Git Standards and Conventions.
4. Put the document into the folder.
5. Push the new file into the Git repository.

1.7 Completing a Github Issue

When an issue is assigned to a team member, the team member is responsible for overseeing its development, or for further decomposing the task into smaller tasks for other members. In the case of the latter, the assigned team member should reference Identifying an Issue or Opportunity. In the case of the former, the following steps should be taken:

1. Ensure a full understanding of the task-at-hand before starting. Ask any questions needed to better understand what is required.
2. Create a design for the software to be developed, and show the design to the owner of the subsystem.
3. Pull the latest changes from the appropriate Git repository.
4. Create a branch off the master branch, and name it according to the Git Standards and Conventions.
5. Update the "README" with a meaningful description of the task undertaken and steps needed to run or test the software.
6. Develop the software using an integrated development environment, and the most recent version of the API and programming languages. Adhere to the Coding Standards and Conventions outlined below.
7. Commit to the local branch whenever a milestone is reached.

8. Once the code is complete, tests should be run to ensure the code reasonably exhibits desired behaviour.

9. Create a pull request for the code to merge it into the master branch, and have it reviewed as per Reviewing Code.

10. Impelment any suggestions by the reviewers.

11. Complete the merge into the master branch, and notify others through a message on Gitter.

12. Mark the Github issue as resolved.

At the end of these steps, the Github issue task should be implemented into the offical Project Imac source code, and the Github issue should be marked as resolved.

1.8 Reviewing Code

When a coding task is complete as per Completing a Github Issue, it should be reviewed by 2 different members of the team who did not participate in resolving the issue. These two team members are randomly selected. The selected reviewers will take the following steps:

1. Ensure a full understanding of the assigned task. Ask any questions needed to understand the scope of the issue.

2. Pull the latest changes from the appropriate Git repository.

3. Test the code to ensure it runs and reasonably behaves as expected.

4. Review the code line by line to look for logical inconsistencies.

5. Review the code line by line to look for inconsistent styling.

6. Provide feedback to the team member assigned to task.

7. Iterate with the team member assigned to the task until the code is sasfactory and the changes have been merged into the master branch.

By the end of these steps, the code should be reasonably consistent with the style guide, and it should reasonably behave as expected.

2 Version Control

Project Imacs will use Git, hosted on Github, for version control. A Github organization will be created to host all of Project Imac's repositories.There will be two repositories within the project: one for storing Project Imac's source code, while another to host documentation. Team members are expected to

concurrently work on their tasks by using separate branches, and then merging their changes back to the master brach by creating a pull request. Pull requests are to be reviewed by at least 2 members before they are merged into the master branch. These 2 members must be different than the team member who wrote the code. Furthermore, Github's Issue Tracking feature will be used tto keep track of tasks, enhancements, and bugs for the projects. The issues created in the issue tracking software will be assigned to individual team members who will be responsible for overseeing the completion of the task.

2.1 Repository Descriptions

This section will provide a description of the repositories used in Project Imacs.

2.1.1 Docs

The `docs` repository is for documents and any research that is done. All notes from research should be uploaded as plain text files in the appropriate `research/` top-level directory. This file should ideally be a Markdown file (`research-foo.md`). Any demonstrations demonstrating the functionality of an algorithm or API should also be located in the `research/` top-level directory.

2.1.2 Dev

The `dev` repository contains the source code for Project Imacs. The code in this repository should adhere to the following Git conventions.

3 Standards and Conventions

3.1 Git Standards and Conventions

3.1.1 Creating and Managing Branches

- Branches are to use the following naming convention: `feature/{Name of the feature implemented}`, `fix/{Name of the fix implemented}`, or `misc/{Name of the task implemented}`.
- Branch names should follow the kebab-case naming convention.
- The master branch is protected, and a pull request must be created in order to merge code into the master branch.

3.1.2 Committing Code

- Do not commit code into another team member's branch without first consulting that team member.
- Always branch from the most recent version of another branch.

- Commit messages should not be more than 50 characters.
- Commit messages should follow the commit.style style guide.
- A commit body is encouraged but optional. Either paragraphs or bullet points in the commit body are acceptable.
- Use `git add -p; git commit -vv` when committing code to carefully review the items to be commited and to provide a good commit message.
- Compiler generated code should not be committed, and their extensions should be included in the Gitignore file. Each team member is expected to generate files from the source on their own machines.
- Do not commit any sensitive information (e.g. student ID numbers) or API keys to any Git repository.
- Do not commit any breaking changes to Git.

Examples of acceptable and unacceptable commit messages are shown below:

What not to do -- Unacceptable

Commit A
```
fixed layout bug
```

Commit B
```
Add file
```

Commit C
```
Small fix
```

Commit D
```
Minor change
```

What you should do -- Acceptable

Commit A
```
Add module to integrate UI with filesystem IO
module

Add a module to allow interfacing with the
filesystem IO from the UI. This allows the UI
code to manage and request data at a much high
level.
```

Commit B
```
Fix typo in documentation for module X
```

Commit C
```
Fix bug causing crashing on start

- Issue occurs because X doesn't do Y
- There is a logical error in the conditionals
for module Y's foo function
- Fixed issue using machine learning algorithms
```

3.2 Coding Standards and Conventions

- 2 space indents should be used for web-development languages such as HTML, CSS, JavaScript and its variants.
- 4 space indents should be used for programming languages not covered in the above bullet point.
- Tabs should exclusively be used in Makefiles.
- JavaScript code and its variants should contain **camelCase** for variables and **SNAKE_CASE** for constants.
- File names should use kebab-case for its file names.
- Line length should not exceed 80 characters.

- Single quotes should be used for strings where applicable.
- All source code files should be prefaced with a comment detailing the purpose of the file, and a high level description of the algorithms.
- All methods and functions should be prefaced with a comment detailing the method's behaviour, required inputs, and expected outputs.
- All global variables should be prefaced with a comment detailing its purpose and usage.

4 Roles and Responsibilities

The following table details the software components and tasks that each team member is responsible for.

Team Member	**Responsible For**
Kelvin Lin	File Management System
Jiahong Dong	File Management System
Liam Casola	Data Visualization
Varun Hooda	High Level Software Design
Mikolaj Hrycko	Data Visualization
Danish Khan	Handwriting & General Input Integration
Prince Sandu	User Experience
Baltej Toor	User Experience

Given that all team members are similarily qualified software engineering students, overlap in resopnsibilities can occur with respect to general software development, software implementation, change validation, testing, and documentation. Team members were assigned responsibility to certain subsystems given their interest towards particular aspects of the system. In the table above, where a person is listed as being responsible for a subsystem, they are expected to take ownership of the system with regards to its design, implementation, testing, and project management. They will be exected to become the most knowledgable person about that subsystem, although all team members are expected to take an active role in designing and implementing all aspects of Project Imacs, regardless of whether or not they are the owners of the subsystem. Team members whose responsibilities are listed as "Software Development" or "Software Testing" are, in addition to the systems that they own, also responsible for assisting with software development and developing test cases in different critical areas on an as-needed basis depending on the progress of the team. Overall, each team member is expected to be well-informed about the overall progress of Project Imacs, while taking an active role in driving the development for their own subsystem.