

Table 1: Revision History

| Date | Developer(s) | Change |
|-------------|---------------------|--|
| Sept. 26 | Pavle Arezina | Added Team Meeting, Communication, Member Roles |
| Sept. 26 | Nicolai Kozel | Added project schedule section |
| Sept. 26 | Riley McGee | Added Proof of Concept Demonstration Plan, Technology, & Git Workflow sections |
| Sept. 29 | Pavle Arezina | Updated Introduction and Formatting |
| Sept. 29 | Nicolai Kozel | Final Proofreading |
| Sept. 30 | Riley McGee | Final additions to Rev 0, formatting corrected throughout |

SE 3XA3: Development Plan Gifitti

Team 2, Gifitti
Pavle Arezina arezinp
Nicolai Kozel kozeln
Riley McGee mcgeer

The Development Plan for Gifitti is to clearly state how the project will be created. The team meeting and communication plan will allow us to work cohesively as a group with clearly defined team member roles. To ensure that the team can work on the project without conflicts, the git workflow plan is defined and the risks pertaining to Gifitti are stated to develop plans to avoid them. The presentation of the code and documentation has been constrained to guidelines to ensure a uniform project is produced on the schedule defined by the group members.

1 Team Meeting Plan

Team meetings will occur on Mondays and Tuesdays. The meetings on Monday will happen twice a day when possible during the lab periods. Tuesday meetings will happen once a day on the first floor of Thode Library. While all three of the group members will contribute during the team meetings, Nick will act as the Team Leader and Riley will scribe the meetings. The agenda will follow the Harvard guidelines. (See Gifitti/ReferenceMaterial/HarvardGuidlines.pdf)

2 Team Communication Plan

Facebook messaging will be utilized to ask simple questions about the project to other group members. To track errors in the documentation or code, GitLab issue tracking will be used to help communicate these errors. A Combination of texting and calling will be used to contact a group member when they cannot be reached through Facebook messaging. Skype is the primary method to conduct calls for team meetings when physically meeting is infeasible. To contact a T.A. or Professor due to questions or issues with the project, mcmaster e-mail will be used.

3 Team Member Roles

Nick will be in charge and will be the Team Leader. His job is to assign tasks and provide a clear goal on what needs to be completed. He is knowledgeable on Gantt charts, git, image manipulation and C#. Riley will be responsible for scribing the team meetings and is experienced with git and C#. Pavle's responsibility will involve the documentation because he is an expert in git and LaTeX.

4 Git Workflow Plan

The development of Gifitti is to follow the Feature Branch git workflow.

4.1 When and how to Create a Feature Branch

- The repository will have no direct code changes made on a local version of master. However all documentation changes and additions may directly be added to master. All development of code is to occur on a feature branch from master, each branch is unique to the feature it addresses. Once a feature is completed and has been reviewed and tested it is to be merged into master. This merge should link the resolution of the issue on GitLab that it solves.
- Feature branch names should follow the following naming practices:
 - Be brief in name of the branch, descriptive in the merge into master after complete
 - Keywords such as fix (for fixing bugs), ui (for view additions), feature (for general additions) followed by a brief of what the branch addresses. i.e. ui/file-menu, fix/console-fault, feature/save-as-sprite etc.
 - As seen in the examples above follow a <Category>/<Brief> with words separated with -
- Commit and push local feature branches as a backup
- Pull requests are not needed for this project. We will all act as admins to the repository and will review feature branches up to date with master as a team before the featurebranch creator merges it into master. This step is only needed for major changes.

4.2 Brief List of Commands for the Workflow

Make a new feature

```
git checkout -b <Category>/<Brief> master
```

Backup Local Feature Branch

```
git push -u origin <Category>/<Brief>
```

Publish Feature into Master

```
git checkout master  
git pull  
git pull <Category>/<Brief>  
git push
```

Committing

All commits are done to the feature branch for code, and master for documentation

4.3 Tags

- All deliverables should be tagged appropriately to mark that it is complete.
- All code releases should be tagged with a version, x.y.z where x is a major update, y is a minor update, and z is a bug fix.

4.4 Milestones

- Mark deliverables and their due dates.
- All milestones are found in the project schedule
- Deliverables are considered major updates to the project.

5 Proof of Concept Demonstration Plan

Risk 1

Reading in GIFs. GIF files must be read into Gifitti and parsed.

Plan for demonstrate risk 1's feasibility

By the project demo an open source library that provides this functionality will be chosen and implemented into a base design.

Risk 2

Sprite Spreadsheet outputs for the currently used gif. This issue is a risk due to it being image manipulation. *A sprite spreadsheet is an image file where animation is broken into frames while stored in one location.

Plan for demonstrate risk 2's feasibility

Image manipulation will be done with an external api. By the project demo this will be chosen. It can be proven that it will be achievable via functionality of the selected api.

Risk 3

The project is only usable on Windows platform, until .NET is made executable on Linux and OSX platforms.

Plan for demonstrate risk 3's feasibility

All demonstrations, and development must be done on a windows platform.

Risk 4

Testing is time consuming for all major components as the application is based on user input and experience.

Plan for demonstrate risk 4's feasibility

Some test cases that can be used to validate the system will be presented during the project demo to show that we can automate most of the testing that does not rely on user interaction. This saves time for the user tests.

6 Technology

Note: Currently, a Windows platform must be used to run and develop this project. In the near future .Net will be ported to OSX and Linux.

6.1 Programming Language(s)

The majority of all development is in C# however WPF will also be used for UI.

6.2 IDE

Visual Studio 2015 Community

6.3 Testing Framework

Visual C# Test Suite, specifically unit test. This platform comes integrated with Visual Studio 2015 Community.

6.4 Document Generation

- Doxygen for code documentation.
- Visual Studio for any post code software model generation
- Gantt Project for the project schedule

7 Coding Style

Coding standard utilized for the project is the .NET Framework Development Guide.

8 Project Schedule

The Gantt Chart outlining the project schedule can be found here.

9 Project Review