Verification and Validation Report: SFWRENG 4G06 - Capstone Design Project

Team #7, Wardens of the Wild
Felix Hurst
Marcos Hernandez-Rivero
BoWen Liu
Andy Liang

October 8, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

[symbols, abbreviations or acronyms – you can reference the SRS tables if needed —SS]

Contents

List of Tables

List of Figures

This document ...

3 Functional Requirements Evaluation

4 Nonfunctional Requirements Evaluation

4.1 Usability

4.2 Performance

4.3 etc.

5 Comparison to Existing Implementation

This section will not be appropriate for every project.

6 Unit Testing

7 Changes Due to Testing

[This section should highlight how feedback from the users and from the supervisor (when one exists) shaped the final product. In particular the feedback from the Rev 0 demo to the supervisor (or to potential users) should be highlighted. —SS]

8 Automated Testing

9 Trace to Requirements

10 Trace to Modules

11 Code Coverage Metrics

References

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

Team

3. Thankfully, our team had no disagreements during this deliverable and were all on the same page, so this will serve as a response to everyone's Q3.

Andy Liang

1. Creating our development plan was crucial for several reasons specific to our ambitious project. First, our game involves complex technical challenges - procedural destructible environments, intelligent slime mold traversal, and physics-based interactions that could compound into performance issues. Without a clear plan, we could easily get lost trying to solve these problems simultaneously. The plan helped us identify our main risk early: ensuring the slime mold behavior works as intended while maintaining performance when combined with our voxel-based destructible environment. By recognizing this upfront, we can focus our proof of concept demonstration on exactly this integration challenge. Additionally, with our diverse team roles (Art Director, Character Artist, Environment Artist, Programmer, Music Director, Composer), coordination is essential. The plan establishes clear communication channels through Discord and GitHub, defines our workflow using pull requests and code reviews, and sets expectations for CI/CD implementation. Without this structure, our different specializations could easily work in isolation and create integration nightmares later. The scheduling aspect also forces us to think realistically about deliverable deadlines and break down our complex technical goals into manageable milestones.

2. Early Issue Detection: Given our concern about compounding errors between procedural systems and physics, automated testing can catch integration problems before they become major headaches. Team Coordination: With multiple people working on different systems (art, code, audio), CI/CD ensures everyone's work integrates properly and nobody breaks someone else's features. Code Quality Assurance: Our plan includes unit testing, security checks, and formatting verification,

which is essential when working with C# and Unit.y Performance Monitoring: Since performance is a key risk with our procedural and physics systems, automated performance testing can flag issues early.

BoWen Liu

1. Creating a development plan prior to starting the project is essential in aligning the team's goal, and workflow in order to have an realistic and feasible starting point and roadmap on how to proceed in this project.

2. CI/CD improves traceability and accountability in one's work both in terms of intra/inter team development as well as for upper management in a business context. The disadvantages to using CI/CD could be low quality of work to meet rigorous and sometimes unrealistic weekly milestones as well as adding unnecessary overhead when committing deliverables.

Felix Hurst

1. Creating a development plan prior to starting a project ensures many aspects of proper organization. Everyone in the team knows what tasks they are responsible for, so different team members do not end up trying to do the same work, and know who to contact to ask questions about specific modules. The team has expectations set, including activity, quality, self-imposed deadlines, and meeting schedules. The team is ultimately guided by the development plan in nearly everything they do while working on the project. Without this kind of structure, team members would be spending a lot more time asking questions, causing delays in development. Or, they may underperform compared to the other team members' internal expectations. It is important that everyone is on the same page to minimize the need for future questions and minimize the possibility of conflict within the team.

2. The advantages of using CI/CD include:

    - Pull requests could be verified to meet specified tests. This ensures poorly written code is not accepted into the repository.
    - New code could be automatically built into a testable version of the project, making it faster to test.

The disadvantages of using CI/CD include:

- It takes time to set it up and write tests, especially those that are intended to be universal across all newly accepted code.
- It may slow down the process of merging pull requests for minor changes that don't need extra testing.

Marcos Hernandez-Rivero

1. Creating a development plan before starting a software engineering group project is essential because it provides a clear roadmap for the team, defining goals, scope, roles, and timelines to keep everyone aligned. It helps prevent confusion, overlap, or missed tasks by assigning responsibilities, establishes coding and documentation standards for consistency, and outlines milestones to manage time effectively. A development plan also anticipates risks to project success, and ideally sets strategies to address them.

2. CI/CD allows teams to integrate code frequently, and catch many errors early and automatically, which in the long run improves software quality and reduces the amount of bugs either on release or later down the production workflow. Some notable disadvantages though, are that it requires setting up and maintaining the CI/CD system, which can be time-consuming and/or confusing to many individuals. Additionally, any automated tests need to be thorough so that they act as a reliable tool to ensure code quality.

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team

able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)