Truthy

Boolean Logic Simulator in C++
Software Development Plan
Version <1.1>

Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>
12/2/2024	1.0	Filled in Roles Section	Evan, Nathan, Darshil, Mitchel, Jay
22/2/2024	1.1	Finished Roles Section and Assigned Sections to each person	Evan, Nathan, Mitchel, Ian

Table of Contents

Software Development Plan

1. Introduction

The *Software Development Plan* outlines information related to the Boolean Logic Simulator in C++. It includes information about the scope of the project, project plan, organization structure and management process and any additional resources used for the project. Additionally, it includes additional resources related to the project including references with explanations of any acronyms, abbreviations and definitions.

1.1 Purpose

The *Software Development Plan* serves as a reference with information relevant to the Boolean Logic Simulator project. It serves to guide the project development, outline key milestones and processes and define goals for the project.

Software Development Plan is used by the following roles:

- The **Project Leader** uses the *Software Development Plan* to guide development and allocate resources.
- The **Quality assurance engineer** uses the *Software Development Plan* to determine what work should be accepted, revised or rewritten.
- All **Implementers** use the *Software Development Plan* to understand the goals, deadlines, and constraints relevant to the project
- **Every team member** uses the *Software Development Plan* to understand their roles, responsibilities and expectations.

1.2 Scope

The *Software Development Plan* describes the Boolean Logic Simulator, the project organization, and the management processes that will be used during the development of the project. It serves as an overall guide for the Boolean Logic Simulator project. The iteration plans provide more concrete details about what work is planned for each iteration. The meeting notes serve as a log of specific topics discussed during meetings and how work is allocated among members after each meeting.

1.3 Definitions, Acronyms, and Abbreviations

The following section provides definitions for any abbreviations, terms and acronyms used in the *Software Development Plan:*

- UPEDU: The Unified Process for EDUcation is a simplified software development process created for educational purposes
- Software development plan: A plan that guides the development of the boolean logic simulator project
- Github: A hosted git solution used for change management and team coordination in software projects

1.4 References

Title	Source	Publishing Organization	Last Updated Date
Vision Document	Has not been created	Truthy	Has not been created
Software Requirements Document	Has not been created	Truthy	Has not been created
Software Architecture Document	Has not been created	Truthy	Has not been created
User Manual	Has not been created	Truthy	Has not been created
Team Meeting and Individual Logs	Project meetings repo	Truthy	2/25/2024

1.5 Overview

This *Software Development Plan* contains the following information:

Project Overview	—	Provides details about the scope, constraints, assumptions, purpose, goals and deliverables for the Boolean Logic Simulator project
Project Organization	—	Describes the structure, roles, and expectations for the Truthy organization
Management Process	—	Provides details about the project plan along with details about specific processes that will be followed while working on the project including requirements management, reporting, risk management and quality control.
Annexes	—	Provides additional resources relevant to the software development plan

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

Project Purpose:

- To develop a program that can evaluate boolean logic and simulate the behavior of logic circuits

Project Scope:

- The program must evaluate AND, OR, NOT, NAND, XOR expressions
- The program must parse through user-input boolean expressions
- The program must correctly handle any errors
    - Mismatched parenthesis
    - Unknown or missing operators
    - Unknown or missing operand
    - Any other issues found
- The program must allow users to define True and False values, represented by T and F

Project Objectives:

- Implement ability to do logical operations AND, OR, NOT, NAND, XOR
- Implement parser to read users-input that correctly handles parenthesis and boolean algebra
- Implement the ability to evaluate entire expressions of boolean operations into a correct True or False value
- Implement an error handler that will be able to correctly handle any issues found
- Develop a accurate set of test cases that will be able to determine the correctness of the boolean logic evaluator by testing many complex logic circuit combinations
- Ensure quality code and readability
- Document various steps of the software engineering process
    - Project Management Plan
    - Requirements Document
    - Design Document
    - Meeting Logs
- Ensure that every artifact or objective is completed on-time

2.2 Assumptions and Constraints

Assumptions:

- Team members will be able to meet every Thursday 4.00 pm,
    - If team members are unable to meet, other platforms of communication are available

- Team members have skills in object oriented programming

Constraints:

- The program must follow object oriented design
- The software engineering process must follow the Unified Process for EDU guidelines
- Must be written in C++
- Project will be due at the end of the semester

2.3 Project Deliverables

Deliverables:

- Project Management Plan (Week 3)
- Software Requirements Document (Week 5)
- Software Architecture Document (Week 10)
- Test Cases (Week 13)
- User Manual (Week 15)
- Program/Source Code (April 30 - May 1)
- Team Meeting and Individual Logs (April 30-May 1)

2.4 Evolution of the Software Development Plan

The *Software Development Plan* will be revised before each Iteration. It may be updated with any revised processes, roles and estimates as needed with the support of the full team between iterations.
(This table will be filled out with new versions as development continues)

Date	Revision Title	Description
<dd/mmm/yy>	<x.x>	<details>
2/25/2024	Initial Submission	The initial version created before the first iteration submitted to canvas

3. Project Organization

3.1 Organizational Structure

The organizational structure of this project team broadly can be grouped into includes Managers, Implementers, and Testers. The roles of these groups are to manage, to write code, and to test the program respectively. Evan, Mitchell, and Ian are all managers. Evan is the project leader, so he is the head of our team and will be the one making the biggest of decisions. Mitchell is the configuration manager so he will provide the environment we will be using and help with infrastructure. Ian is the change control manager, so he will be reviewing the code and informing us of changes to make. Everyone in the group is an implementer meaning they will be writing code in the provided environment of our configuration manager, Mitchell, and reviewing and changing our code based on the decisions of the change control manager, Ian. Evan and Jay are our testers, so they write automated tests to ensure different components of the software project works correctly.

3.2 External Interfaces

3.3 Roles and Responsibilities

Person	Unified Process for EDUcation Role
Evan Almloff	Project Leader, Implementer, Tester, Integrator, Stakeholder
Jay Chung	Implementer, Reviewer, Tester
Mitchell Leonard	Analyst, Implementer, Configuration Manager, Quality Assurance Engineer 2
Nathan Abraham	Designer, Implementer, Quality assurance engineer, Notetaker
Darshil Patel	Implementer, Meetings coordinator
Ian Lim	Implementer, Change Control Manager, Stakeholder

All team members can perform Any Role activities

4. Management Process

The management process for the Boolean Logic Simulator project will be guided by the Unified Process for Education (UPEDU) framework, emphasizing a structured yet adaptable development methodology. This approach is characterized by its iterative and incremental nature, where the project is divided into distinct phases: Inception, Elaboration, Construction, and Transition. Each phase focuses on specific project goals and deliverables, with the project evolving through collaboration among specialized roles within the team.

- Project Leader: Evan Almloff will oversee the project coordination, ensuring that the project meets its objectives and deadlines. Evan will also contribute as Implementer, Tester, Integrator, and act as a Stakeholder.
- Team Roles: Each team member has been assigned specific roles, ensuring a balanced distribution of tasks according to their expertise.

4.1 Project Estimates

- The estimated schedule will be developed based on initial assessments of the project scope and complexity. This will include developer hours, resources needed, and potential software requirements.
- Estimated Schedule: The project is estimated to take 2 & 1/2 months from now to the first release.
- Basis for Estimates: Based on the complexity of the project, historical data from similar projects from previous semester projects, and initial requirement analysis.
- Re-estimation Points: After the completion of each major phase (design, implementation, testing), and in response to significant requirement changes or unforeseen challenges.

4.2 Project Plan

- Schedule and Resources: Detailed in the Phase Plan and Project Schedule sections.

- Artifact and Iteration Schedules: Artifacts include management plans, requirements documentation, design documents, implementation code, test cases, and user manuals. Iterations are aligned with project parts assigned on specified weeks.

4.2.1 Phase Plan

4.2.2 Iteration Objectives

Each phase will consist of multiple iterations, with the following objectives:

- Requirements Analysis: Gather requirements, define the scope, and understand user needs.

- Design: Develop architecture, select technologies, and create detailed design documents.
- Implementation: Code development, unit testing, and integration of Boolean logic simulation functionalities.
- Testing & Deployment: System testing, bug fixing, user acceptance testing (UAT), and deployment.

4.2.3 Releases

- Alpha Release: A prototype demonstrating basic Boolean logic operations, for internal review. Implementation deadline.
- Beta Release: A feature-complete version for external testing, incorporating feedback from the alpha phase. Test cases.
- Final Release: The fully tested and bug-free version, ready for public use. Final submission.

4.2.4 Project Schedule

Major Milestones:

- Project Management Plan
- Project Requirements
- Project Architecture and Design
- Project Implementation
- Project Test Cases
- Project User Manual

Deadlines:

- Task 1: Management Plan due Week 3
- Task 2: Requirements due Week 5
- Task 3: Design due Week 10
- Task 4: Implementation due Week 11
- Task 5: Test Cases due Week 13
- Task 6: User Manual due Week 15
- Final Submission Prep: April 30 - May 1

	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
Task 1														
Task 2														
Task 3														
Task4														
Task 5														
Task 6														
Final														

4.2.5 Project Resourcing

- Staff Requirements: The team consists of six members, each with defined roles. Additional resources may include the help of the Professor and GTA's, especially in areas requiring specialized knowledge.
- Training: Specific training requirements will be identified based on the project needs. Target dates for training completion will be set based on the project schedule, ensuring team members are prepared for their roles.

4.3 Project Monitoring and Control

- There are 6 key features (Requirements) that will measure our progress, as those are the requirements we must complete. We will report every time a requirement has been completed. No changes are allowed to our product requirements.
- The method we will use to control the quality of our project will be a unified one. When errors are found, corrective maintenance is needed, and we will follow our risk management plan. Once the project has all of the requirements, we will do perfective maintenance which will include: Adding descriptive comments, Adding details that are not required (truth table), and condensing and removing unnecessary code where need be.
- We will take a unified approach where we test our code frequently to identify risks, and incrementally and iteratively add to the code, without making changes to our requirements.
- The process we will take to submit changes is simple: When an individual of the group finds a problem in the code, he will edit a file and submit a request, where the change control manager will accept or deny the specific change in github.

4.4 Requirements Management

The requirements for this system are captured in the Vision document. Requested changes to requirements are captured in Change Requests, and are approved as part of the Configuration Management process.

4.5 Quality Control

For documents and records, final submissions will be handed over to quality assurance managers for review. Suggestions and corrections will be tracked through comments in the shared google doc for each document. The quality assurance managers will update the revision history above to reflect these changes.

For code deliverables, quality assurance managers will conduct a review process to guarantee functionality, readability, consistency, and adherence with project guidelines. For any issues discovered by team members, testers, or quality assurance managers during the review process, a pull request will be submitted on the project code repository. Quality assurance managers will review the issues and alert the Change Control Manager who will deny or approve after consideration. In either case the project lead will be notified of the decision when appropriate and in the case of approval, the change will also be documented.

4.6 Reporting and Measurement

4.7 Risk Management

Risks will be identified through team discussion and review. It is important to complete this step in the planning phase of the project so the team has an understanding of what to be cautious of moving forward. Revaluation of the risks and constraints will happen when the team feels an appropriate concern is raised.

4.8 Configuration Management

Configuration manager will be in charge of choosing tools, standards, and environments to accomplish the project components in a manner that is consistent among team members. Changes and suggestions will be recorded according to the deliverable. For documentation and records, the google doc comment boxes will

be used to communicate and document. For code deliverables, changes and suggestions will be recorded in the pull requests of the code repository. All code, documentation, and relevant materials will be consolidated and stored. Final deliverables will be formatted and prepared for submission with necessary components according to the standard the configuration manager set.

5. Annexes

The annexes section serves as a comprehensive repository of additional materials that are pertinent to the reader of the Software Development Plan. This section aims to enhance understanding and ensure the alignment of the project development with established technical standards and guidelines. Below is an outline of the annexes relevant to the Boolean Logic Simulator project in C++.

1. Unified Process for Education (UPEDU)

- Overview: A tailored version of the Unified Process specifically designed for educational settings. UPEDU emphasizes iterative development, risk management, and student learning outcomes.
- Application to Project: The project will adhere to the UPEDU framework, ensuring a structured and phased approach to development, with specific emphasis on iterative feedback and risk assessment.

2. Programming Guidelines

- Coding Standards: The project will follow a set of predefined coding standards to ensure consistency, readability, and maintainability of the C++ codebase. This includes naming conventions, comment and documentation practices, and code structuring.
- Version Control: Guidelines for the use of version control systems to manage changes and collaboration among team members. This includes commit messages, branching strategies, and merge policies.

3. Design Guidelines

- Architecture Design: Principles and practices for designing the software architecture of the Boolean Logic Simulator, ensuring scalability, performance, and modularity.
- User Interface Design: Guidelines for the development of user-friendly interfaces, considering usability principles and accessibility standards.

4. Quality Assurance Guidelines

- Testing Standards: Detailed processes and techniques for thorough testing of the software, including unit testing, integration testing, and system testing. This ensures the reliability and correctness of the simulator.
- Review Processes: Procedures for code reviews and design assessments to maintain quality and adherence to project standards throughout the development lifecycle.

5. Process Guidelines

- Iteration Planning: Detailed guidelines for planning and executing iterations within the project, including objective setting, task allocation, and progress tracking.
- Risk Management: Strategies for identifying, assessing, and mitigating risks throughout the project development process.

References Section

- This section includes references to external documents, tools, and resources that provide additional context or support for the guidelines and standards mentioned above. Examples include official documentation for C++ programming, UPEDU framework resources, and software development best practices.

Conclusion

- The annexes section ensures that all project team members have access to critical guidelines and standards that govern the development of the Boolean Logic Simulator. By adhering to these directives, the team aims to achieve a high-quality, efficient, and effective software development process, resulting in a robust and user-friendly simulator.