**Chocaholics Anonymous**

***SOFTWARE PROJECT MANAGEMENT PLAN (SPMP)***

**Version No. 1.0**

NOTE: THIS IS A PLAN AND WOULD BE REVISED OFTEN AND TURNED IN as part of CHECK-IN (e.g., new estimates, timelines, risks, etc.). The Risk table here would match the other Risk Document you create. TRACEABILITY!

Project Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| VersionNumber | Date | Revision Author | Description of Revision |
| 1.0 | 6/12/2023 | Team 2 | Original Published Version |
|  |  |  |  |

|  |
| --- |
| **Table of Contents** |

[1. Introduction 4](#_Toc127683410)

[2. Project Estimates 6](#_Toc127683413)

…

**1.0 Introduction**

The Software Project Management Plan outlines the project problem, cost and time estimates, risks and management approach, schedule, staff organization, and the tracking and control mechanisms for ChocAn organization. The primary goal of this project is to address the challenges that hinders ChocAn's organization processes and deliver automated and efficient data processing software. This software will improve member validation, streamline billing and record-keeping processes, provide a comprehensive provider directory, and establish real-time integration with Acme Accounting Services. ChocAn’s overall efficiency, accuracy, and service delivery will be improved upon the successful completion of this project.

* 1. **Problem statement**

ChocAn is encountering several critical challenges in its current system, which sets back its overall efficiency and accuracy. Manual member validation results in inefficiencies and delays in delivering timely services. Inadequate record-keeping and billing processes lead to inconsistencies and errors in financial reporting and reimbursement to providers. Limited access to a comprehensive provider directory creates a strain for providers in accurately identifying and selecting appropriate service codes. Additionally, the lack of real-time integration with Acme Accounting Services for membership fee processing causes delays in suspending or reinstating members based on their payment status.

* 1. **Project scope**

The project scope includes developing and implementing a new data processing software for ChocAn. The software will address the identified challenges and deliver the following functionalities in a priority sequence:

1. Member Validation:
   * Automation of member eligibility verification.
   * Real-time access to member information.
   * Instant validation of member status.
2. Billing and Record-Keeping:
   * Centralized billing system.
   * Automated recording of service details.
   * Streamlined billing processes.
   * Comprehensive financial reporting.
   * Timely reimbursement to providers.
   * Efficient tracking of services provided.
3. Provider Directory:
   * Easily accessible and updated Provider Directory.
   * Accurate service codes and detailed service descriptions.
   * Associated fees for each service.
   * Regular updates for changes or additions.
4. Integration with Acme Accounting Services:
   * Real-time updates on membership fee payments.
   * Timely suspension or reinstatement of members based on payment status.
   * Elimination of inconsistencies in membership management.

**1.2.1 Inclusions**

The software will include modules for member validation, billing, record-keeping, Provider Directory management, and integration with Acme Accounting Services. It will also have interfaces for providers to access and update member records (add comments), submit service details, and view reports. The software will be designed to handle high volumes of data efficiently and maintain data integrity.

**1.2.2 Exclusions**

The following aspects are excluded from the project scope:

* Design and development of communications software and provider’s terminal.
* Implementation of the EFT component for financial transactions.
* Sending files as email attachments (reports will be generated as separate files).
* Any modifications to Acme Accounting Services’ internal processes.

**1.3 Major software functions**

The software functions will be divided into the following major modules:

* Member Validation: Instant verification of member eligibility and status.
* Billing and Record-Keeping: Automated processing and recording of service details and fees.
* Provider Directory: Comprehensive and easily accessible service code directory.
* Acme Accounting Services Integration: Real-time updates on membership fee payments.

**1.4 Performance/Behavior constraints**

The software must meet the following performance and behavior constraints:

* Real-time response for member validation and billing processes. The system has minimal timeliness requirements, i.e. it is acceptable if operations take a few seconds. However, operations must be completed in the order they were sent.
* Batch processing for end-of-week accounting procedures should be available to simplify operations.
* The system should be scalable to handle a large number of ChocAn members and providers efficiently.
* All financial transactions must be accurate and secure.
* The software should be user-friendly and intuitive for providers to use efficiently.
* Provider interfaces should be accessible from multiple terminals simultaneously.

**1.5 Management and technical constraints**

**1.5.1 Management constraints**

* The completion of the project will be done by 5 people and no additional personnel are available to be utilized for the project.
* The software planning and scoping must be completed by August 22nd 2023.

**1.5.2 Technical constraints**

* This software project must utilize a programming language that is capable of handling complex data operations, e.g., Python.
* A Relational Database Management System will be used, e.g., PostgreSQL.

**2.0 Project Estimates**

**2.1 Historical data used for estimates**

The estimations of documentation deliverables are based on prior experiences of team members on similarly sized class projects. The software code development estimates are based on small and medium sized software development projects completed within work and independent prior projects.

**2.2 Initial Estimate**

**2.2.1 SPMP Completion Estimate**

Estimated time to complete the SPMP for the ChocAn project is 18.5 person hours. Expecting 1 hour of learning / planning of the SPMP components per person total of 5 people prior to beginning the document. 1 hour and 2 people completing initial information from prior knowledge. 1 hour and 5 people of collaborative development of the SPMP for a total of 5 person hours. 1 hour and 4 people completing detailed review of SPMP sections 3 thru 6. Concluding with 0.5 hours and 5 people performing a final review and approval of the SPMP.

In total the SPMP actually took a total of 22 person hours to complete the documentation including planning, executing and reviewing both individually and as a team.

**2.2.2 Overall project estimate**

**2.2.2.1 Line-of Code Estimate**

The lines of code required to meet the project needs above is estimated at 5,000 lines of code. Using a lines of code of 20 lines of code per 8 hour day per person the total time to complete the project would be 2,000 person hours.

**2.2.2.2 Function Estimate**

Separating the functions required into categories of external inputs, external outputs, external inquiries, external interface files and internal logical files results in the below functions for each category.

* External inputs (EI): user, provider, service record data of fees, cost and dates.
* External outputs (EO): Membership status, service cost lookup, service record data, member service weekly reports, provider service weekly reports.
* External inquiries (EQ):provider total services, subscriber account status.
* External interface files (EIF): provider total fee.
* Internal logical files (ILF): user data, member account data, provider specialty, service code name and cost data, and service records.

Table 1: Component complexity factors

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Complexity | | |
| Low | Average | High |
| EI | 3 | 4 | 6 |
| EO | 4 | 5 | 7 |
| EQ | 3 | 5 | 6 |
| EIF | 5 | 7 | 10 |
| ILF | 7 | 10 | 15 |

For the functions within the project EI and EIF are average complexity and EQ, EO, EQ and ILF are high complexity.

Table 2: Project function quantity and complexity.

|  |  |  |
| --- | --- | --- |
| Category | Quantity | Complexity factor |
| EI | 5 | 4 |
| EO | 5 | 7 |
| EQ | 2 | 6 |
| EIF | 1 | 7 |
| ILF | 5 | 15 |

5\*4+5\*7+2\*6+1\*7+5\*15 = 149 function points

Using the function point estimate of 8 hours per function point the total estimate is 1192 person hours.

**2.2.2.3 Tasks Estimate**

The tasks required to complete the project scope and estimate time to complete in hours are:

* Software project management plan - 18.5 person hours.
* Build ERD - 50 person hours.
* Business analysis - 30 person hours.
* Initial code development - 1000 person hours.
* Code reviews - 200 person hours.
* Traceability validation - 100 person hours.
* Unit testing - 250 person hours.
* System testing - 250 person hours.
* System integration testing - 250 person hours.
* User training - 100 person hours.
* Deployment - 200 person hours.
* Final acceptance validation - 100 person hours.

Total task based time estimate of 2548.5 person hours.

**2.2.2.4 Total overall project time estimate in hours of effort**

Based on the three above estimates using lines of code, function based and task based project estimates the total estimate for the project is 2250 person hours. For this project the function block estimate does not appear to accurately reflect the whole project but rather the code development effort. The task based estimate is accounting for a more complete scope of work for the project.

**2.3 Estimation techniques applied and results**

**2.3.1 Estimation technique 1 – lines of code**

To estimate the lines of code required to complete the project the functionality of the software was broken down into its 4 parts and the code required to coordinate the functions. Estimating for each section of code results in the following estimate:

Table 3: Lines of code estimate breakdown

|  |  |
| --- | --- |
| Function | Estimated lines of code |
| Main functionality and coordination | 2,500 |
| Member validation records and verification | 1,000 |
| Billing and record keeping | 1,200 |
| Provider directory | 800 |
| User account management | 1,500 |
| Total | 7,000 |

**2.3.2 Estimate for technique 1 – lines of code**

Using the updated 7,000 lines of code and estimated 20 lines of code per 8 hour day per person, the total time to complete the project is estimated at 2,800 person hours.

**2.3.3 Estimation technique 2 – function points**

Separating the functions required into categories of external inputs, external outputs, external inquiries, external interface files and internal logical files results in the below functions for each category.

* External inputs (EI): user, provider, service record data of fees, cost and dates.
* External outputs (EO): Membership status, service cost lookup, service record data, member service weekly reports, provider service weekly reports.
* External inquiries (EQ):provider total services, subscriber account status.
* External interface files (EIF): provider total fee.
* Internal logical files (ILF): user data, member account data, provider specialty, service code name and cost data, and service records.

Table 4: Component complexity factors

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Complexity | | |
| Low | Average | High |
| EI | 3 | 4 | 6 |
| EO | 4 | 5 | 7 |
| EQ | 3 | 5 | 6 |
| EIF | 5 | 7 | 10 |
| ILF | 7 | 10 | 15 |

For the functions within the project EI and EIF are average complexity and EQ, EO, EQ and ILF are high complexity.

Table 5: Project function quantity and complexity.

|  |  |  |
| --- | --- | --- |
| Category | Quantity | Complexity factor |
| EI | 5 | 4 |
| EO | 5 | 7 |
| EQ | 2 | 6 |
| EIF | 1 | 7 |
| ILF | 5 | 15 |

5\*4+5\*7+2\*6+1\*7+5\*15 = 149 function points

Using the function point estimate of 8 hours per function point the total estimate is 1192 person hours.

**2.3.4 Estimation technique 3 – process/task**

Breaking the project into individual tasks allowed for more clear allocation of time and resources to various tasks to complete the project and better assessment of the project as a whole.

**2.3.5 Estimate for technique 3 – process/task**

* Software project management plan - 40 person hours.
* Build ERD - 50 person hours.
* Business analysis - 30 person hours.
* Requirement gathering - 80 person hours.
* Requirement creation - 80 person hours.
* Requirement validation - 40 person hours.
* Initial code development - 1000 person hours.
* Database creation - 80 person hours.
* Code reviews - 200 person hours.
* Traceability validation - 100 person hours.
* Unit testing - 250 person hours.
* System testing - 250 person hours.
* System integration testing - 250 person hours.
* User training - 100 person hours.
* Deployment - 200 person hours.
* Final acceptance validation - 100 person hours.

Total task based time estimate of 2,850 person hours.

**2.4 Reconciled Estimate**

Based on the three estimation techniques we are estimating the total project to take 2,800 person hours. This is based primarily on the process/task estimate and backed up by the lines of code estimate which are close estimates to each other after incorporation of all parts of the project plan. The function point estimate is significantly lower than the other estimates. We are choosing to not use the function point estimate as it assumes much faster development time for each function point then is feasible given the requirement gathering and validation needed for the functions.

**2.5 Project Resources**

In order to complete this project there are required five resources for the project team, development hardware to simulate the production environment and run test cases. There are gitlab repositories and visual studio licenses for developing the software. GTest for running testing frameworks. Also will require various management tools including GitLabs and Google services.

**3.0 Risk Management**

**3.1 Project Risk Table**

Table 6: Project Risk Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Description | Probability | Impact severity | Impact | Mitigation Plan | Contingency Plan |
| Technical requirements outside team experience | Low | Med | Potential design failures or implementation bugs | Proper planning and requirement gathering should highlight any gaps, early enough to avoid issues | If needed pull resources from other teams, otherwise use consultants if the needed skill set does not exist in company |
| Developer capacity | Low | Med | Overworked developers, leads to higher attrition rate, greater number of bug in implementation, and skipping documentation and testing | Get estimates from developers early. Compensate estimates to be pessimistic as uncertainty is at its highest for these estimates. Management must review and update capacity planning as certainty changes | Pull resources from other teams |
| Security vulnerabilities | Med | Low | User data could be leaked, which would create legal issues for customers. Potentially violate our contract terms | Have cyber security analysis done to identify needed steps early in development | Use a task force to understand what data was leaked and how. communicate with customer the scope of the breach |
| Missing/delayed supplied frontend terminal software needed for validation | Med | Med | Would delay final release of software due to delay of integration testing | Active supplier management plan, frequent status updates | Communicate to customers that the chosen supplier will be delaying the project. Create dummy programs to use for testing |
| Missing/delayed supplied payment software needed for validation | Med | Med | Would delay final release of software due to delay of integration testing | Active supplier management plan, frequent status updates | Communicate to customers that the customer their chosen supplier will be delaying the project. Create dummy programs to use for testing |
| Lose key team members | Med | Low | Could create technical gaps impacting project timing and/or software quality | Maintain documentation to prevent knowledge loss, identify areas where single individuals hold key expertise | If needed pull resources from other teams, or use consultants if skill set does not exist in company |
| Inaccurate Software Project Plan | High | Med | Delays to project, missing needed content | Scheduled reviews and updates to the plan. As certainty changes the plan should reflect the new information | Pull additional resources needed to make up for time loss. Determine if any features can be delivered in the maintenance period to decrease project scope |
| Changes in requirements during development | High | Med | Could create expensive reworks, undermining work done on design, architecture, implementation. Could compromise SW quality | Effective requirements gathering and creation. Enforced traceability from project plan and requirement gathering. a focus on active and effective communication with the customer | If change was driven by customers, explain the impacts of the change and either decrease scope or increase resources. If change was driven by internal failure, determine if any features of the software can be delivered in the maintenance phase. If all else fails find more resources internally |
| Missing documentation | High | Low | Could make software maintenance more expensive. Increases risk from employee turnover | Active supplier management plan. Get frequent status updates from supplier | Use developer time to determine the information needed from the implementation |
| Missing unit testing | High | Low | Potential bugs in deployed software. Could lead to more expensive reworks if bugs are caught in a later stage of testing | Ensure requirements created are testable. When creating requirements have an attribute detailing the plan how the requirement will be tested against the implementation. Ensure time is planned and protected. | Frequent system testing will ensure requirements are fulfilled |
| Scalability / performance issues | Low | High | Software would fail to meet customer needs in use. Requiring expensive rework | Through reviews in the design phase. Planning for use case and potential changes in use case | Dedicated developer time to improve performance |
| Provided databases delivered late | Med | Med | Would delay final release of software due to delay of integration testing | Active supplier management plan, frequent status updates | Create dummy programs to use for testing |

**3.2 Overview of Risk Mitigation, Monitoring, Management**

Twice weekly the status of all identified risks must be reviewed by project management. In these meetings updates will be presented and the current status of each risk updated. Risks will be tracked using green yellow red statuses. As risks become more likely to happen the mitigation plan must be reviewed. In this meeting the team will determine when contingency plans should be implemented.

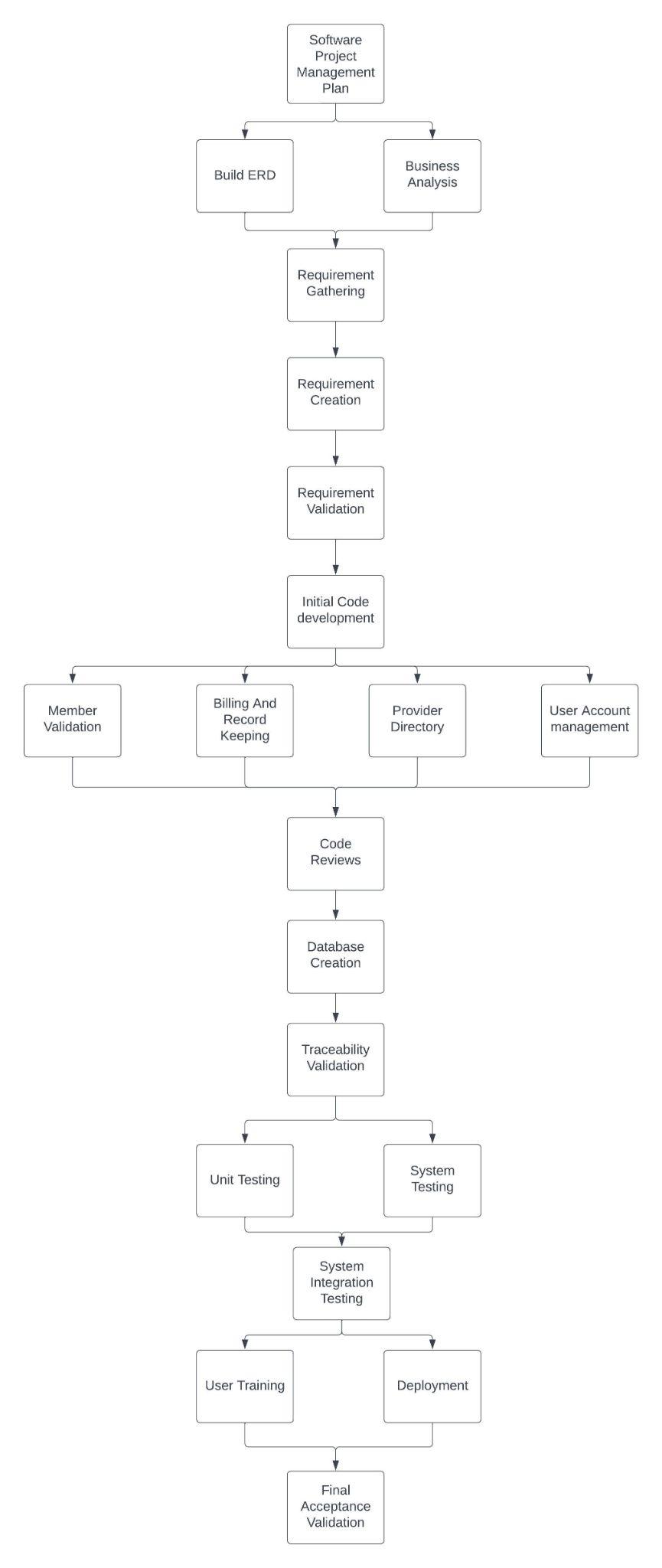
**4.0 Project Schedule**

**4.1 Project task set**

* **Software project management plan**: This task involves creating a detailed plan that outlines the project's objectives, scope, schedule, budget, resources, and risk management approach. It acts as a roadmap for the project's execution.
* **Build ERD**: An ERD is a visual representation of the database schema that shows how different entities in the system are related to each other. Building the ERD is essential for designing the database structure.
* **Business analysis**: This task involves understanding and analyzing the business requirements, processes, and objectives of the organization to ensure that the software aligns with their needs.
* **Requirement gathering**: In this phase, the project team collects detailed functional and non-functional requirements from stakeholders, users, and other sources to define what the software should do.
* **Requirement creation**: After gathering the requirements, they need to be documented and transformed into a clear and concise set of functional specifications for the development team to follow.
* **Requirement validation**: The gathered requirements and specifications are reviewed and validated to ensure they are complete, consistent, and accurate.
* **Initial code development**: This task involves the actual coding and programming of the software based on the validated requirements and design.
* **Member validation**: This could refer to a validation process where the members of the project team review and validate each other's work to ensure quality and correctness.
* **Billing and record keeping**: This task might be related to implementing features or functionalities related to billing and record-keeping in the software.
* **Provider directory**: Developing a directory of service providers or vendors could be a specific requirement for the software project.
* **User Account management**: This task involves creating features for user registration, login, and managing user accounts within the system.
* **Database creation**: After building the ERD, the actual database needs to be created and set up to store the application's data.
* **Code reviews**: Regular code reviews are conducted to identify and fix any coding issues, ensure code quality, and promote best practices.
* **Traceability validation**: Ensuring that each requirement is traceable to the corresponding design and code, and vice versa, to maintain consistency and fulfill all specified requirements.
* **Unit testing**: Testing individual units or components of the software to validate that they work as expected and meet their specifications.
* **System testing**: Conducting tests on the integrated system to ensure that all components work together correctly and meet the overall requirements.
* **System integration testing**: Testing the software in an integrated environment to validate its compatibility with external systems and dependencies.
* **User training**: Preparing and conducting training sessions for end-users to familiarize them with the software and its functionalities.
* **Deployment**: This involves deploying the software to the production environment for actual use.
* **Final acceptance validation**: The project team and stakeholders perform a final acceptance validation to ensure that all requirements have been met, and the software is ready for release.

**4.2 Task network**

Task network is shown below using vertical from top to bottom over time. Horizontally aligned tasks are performed at the same time.



**4.3 Timeline chart**

The timeline chart illustrates the schedule for the project, showing when each task is planned to start and finish. It may include a timeline for the entire project, displaying the overall project duration.

Table 7: Project Task View

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Duration (Business Hour) | Start Date | End Date |
| Software Project Management Plan | 40 | 2023/June/29 | 2023/July/06 |
| Build ERD | 50 | 2023/July/07 | 2023/July/17 |
| Business Analysis | 30 | 2023/July/18 | 2023/July/26 |
| Requirement Gathering | 80 | 2023/July/27 | 2023/August/09 |
| Requirement Creation | 80 | 2023/August/10 | 2023/August/23 |
| Requirement Validation | 40 | 2023/August/24 | 2023/August/30 |
| Initial Code Development | 1000 | 2023/September/04 | 2024/February/07 |
| Database creation | 80 | 2024/February/08 | 2024/February/21 |
| Code Reviews | 200 | 2024/February/22 | 2024/April/02 |
| Traceability Validation | 100 | 2024/April/03 | 2024/April/19 |
| Unit Testing | 250 | 2024/April/22 | 2024/June/07 |
| System Testing | 250 | 2024/June/10 | 2024/July/26 |
| System Integration Testing | 250 | 2024/July/29 | 2024/September/13 |
| User Training | 100 | 2024/September/16 | 2024/October/03 |
| Deployment | 200 | 2024/October/04 | 2024/November/14 |
| Final Acceptance Validation | 100 | 2024/November/15 | 2024/December/03 |
| Total | 2850 | 2023/June/29 | 2024/December/03 |

**5.0 Staff Organization**

**5.1 Team structure**

The team consists of five members, each with their own responsibilities. Cross checks are designed such that no important decisions can be made by one single member of the team.

Members and roles are as follows:

* Samar Alkhalifah is responsible for defining software requirements and verification (i.e., ensuring that the requirements achieve the end goal). This includes communication with the customer to develop suitable requirements.
* Bill Crowe is the project manager, responsible for ensuring that planning is comprehensive, and later that the project moves according to plan. Bill will be cross checked by the team leader of the most relevant team.
* Trevor Gross is responsible for hardware and software integration, including planning for deployment and ensuring that CI infrastructure is functional.
* Abhinav Mukthavaram is in charge of project validation (i.e., ensuring the project meets the requirements). He will be heavily involved with the quality assurance process. Abhinav and Samar will cross check each other’s work.
* Maxwell Petit is the lead software developer and will be responsible for managing developers and overseeing the technical aspects of the project.

**5.2 Management reporting and communication**

The team defines a few classes of communication that will be followed:

* High level product discussion and communication with the customer: this must be an easily archivable format, so email has been chosen. Any serious discussion related to project requirements should be done here.
* Short informal group and one-on-one communication: lower priority communication can take place via an IM service. WhatsApp has been selected for this purpose.
* Requirement design: requirements will be iterated on and tracked in accordance with our GitLab control mechanisms, elaborated on in section 6.2. Archive versions of requirements documents will be held in Google Drive and shared with the customer via email.
* Progress reporting: progress status will be tracked via GitLab devops tools. Report summaries based on this information will be sent to the customer via email when requested.

**6.0 Tracking and Control Mechanisms**

**6.1 Quality assurance and control**

The team will use a combination of tools to ensure that working code is kept functional, and that produced code will meet the customer’s expectations. These include:

* Unit testing: our team will make use of unit testing frameworks to assist developers in producing locally correct code. Unit tests will be used to the extent possible, employing mocks to avoid reliance on a full stack.
* Integration testing: our code will make use of doctests (for public API) and database-backed integration tests. Fuzzing and parameter testing will be employed where reasonable.
* Frontend testing framework and hand evaluation: GUI testing is possible to some extent (with frameworks such as Jest), and this automated testing will be employed where possible. There will also be a degree of hand evaluation: inconsistencies here will be reported in issues.
* Verification against customer requirements and output validation against project requirements: this will be done by hand and under Abhinav and Alkhalifah’s guidance. Results will be tracked in “Verification” and “Validation” GitLab epics.
* Our infrastructure team will provide the customer with a staging instance that is updated at regular intervals. Expectations for these instances will be clearly communicated to the customer, but it will provide a straightforward way to learn about and correct requirement errors as they develop.

All testing that can be run during continuous integration (CI) will be, to minimize downtime due to test failure. This includes the unit and integration testing, as well as a some degree of GUI testing. Failures will be reported on our GitLab instance to allow team members to easily reason about code correctness.

**6.2 Change management and control**

Our team separates two classes of tracked changes:

* Git for SCM: Where possible, we will employ Git to track file history. All source code will be held in our GitLab instance. We will use branch workflow for mainline development, and tags to represent functional points (e.g. where code may be deployable to the staging image)
* Google Drive for DMS: For any types of files not suitable for tracking via Git, Google Drive will be used as a form of document management system. In accordance with our team’s Communication Plan, there will always be a presentable release version of a file available. Google Drive tracks file history, so an archive naturally exists.

**6.3 Tools**

Many of the tools our team intends to use have already been listed in the above sections. The below enumerates these tools and provides a summary of their uses:

* Git SCM: Git will be used to track source code and streamline development. The upstream will exist on our GitLab instance, discussed in the next point.
* Gitlab DevOps: we will use GitLab to track a number of things:
  + Git history: this instance will provide the upstream for our SCM
  + CI pipeline: we will use continuous integration to ensure that our code introduces no new bugs and meetings agreed upon coding standards.
  + Code review: the merge request interface provides a straightforward way to review and approve code. All code will need team approval before being merged, and the interface allows for a way to enforce this.
  + Requirements & Design: the output of requirements and design processes will mostly live in Google Drive. In GitLab, we track the progress of these in epics, which are used to indicate business scope (one epic for Requirements and one for Design). Issues will be used to show deliverables, and labels such as “proposed”, “accepted”, “awaiting customer” will indicate progress.
  + QA tracking: like requirements, the verification and validation stages will also be tracked in epics. Feedback from the QA stage will propagate via properly categorized issues to the correct area.
  + Issues: issues represent units of work to be completed. They will be assigned to an epic (i.e. a stage scope, such as “Requirements”), a milestone (a unit deliverable), and potentially an iteration (a specific scope of time to be completed).
  + CD: continuous deployment will be used to keep the staging server up to date. Later, this will also be used for the production environment.
* Google Drive DMS: Our Google Drive will act as a DMS, and follow the versioning scheme presented in the Team Communication Plan. This will track all files that are not suitable for Git, e.g. PDF or document processor documents. Documentation will have tracking as they move through the stages of “draft” to “accepted”. This is needed to ensure traceability from customer requests through to validation stages.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Review and Signoff of the SPMP | | | | |
| Name | Project Team Role | Signature | Date |
|  | Business Analyst |  |  |
|  | Systems Analyst |  |  |
|  | ….. |  |  |