SOFTWARE DEVELOPMENT PLAN

PROTECT MILITARY BASES

Prepared by:

Hoda Dehghanisanij
Kevin Nguyen
Riley Webber
Louis Zuckerman

CSU Fullerton, CA
11/04/2020

TABLE OF CONTENTS

List of Figures

List of Tables

Revision History

Version	Date	Author	Rationale
PMB 1.0	10/26/2020	Hoda	Created an Outline Structure for SDP
PMB 1.1 & 1.2	10/27/2020	Hoda	Defined Vision/Reference Document for SDP
PMB 1.3	10/29/2020	Hoda	Defined Process Model, Workflow and Role Diagrams
PMB 1.4	10/30/2020	Riley	Gantt Chart, Project Development Scheduling
PMB 1.5	10/30/2020	Kevin	Project Estimations
PMB 1.6	10/30/2020	Hoda	General Software Development Activities
PMB 1.7	10/31/2020	Louis	Elaborating Development Plan
PMB 1.8	11/01/2020	Kevin	Defined COCOMO
PMB 1.9	11/01/2020	Hoda	Defined TR & Management Review
PMB 2.0	11/02/2020	Hoda	Defined Risk Management
PMB 2.1	11/02/2020	Riley	Defined Software Environment and Infrastructure
PMB 2.2	11/02/2020	Riley	Defined System and Software Requirements
PMB 2.3	11/03/2020	Louis	Unit Integration and Testing

PMB 2.4	11/04/2020	Hoda	Defined Software Design
PMB 2.5	11/04/2020	Riley	Defined Software QA and Implementation Phase
PMB 2.6	11/04/2020	Kevin	Defined Requirement Fulfilment
PMB 2.7	11/4/2020	Louis	Review and Format

SECTION 1. INTRODUCTION

1.1 SCOPE

This Software Development Plan (SDP) establishes the structured plan (Software Life Cycle) for software requirements, design, implementation, test, and qualification for the Protect Military Bases (PMB) Computer Software Configuration Items (CSCIs). It also includes modification, reuse, maintenance and other activities necessary in a deliverable software product.

Updates to this SDP will address future PMB software upgrades.
The benefits the system design offers will be considered through the lens of the United States Border Patrol such as reduced need for manned units, increased interception times, robust environmentally resistant hardware etc.

1.2 PURPOSE

The main objective of this SDP is to identify the requirements and standards of the PMB software. This document shall demonstrate the tools for monitoring software development, the processes and methods to be followed for software development, the approach to be followed for each task, and project schedules, organization, and resources for the PMB system.

1.3 SYSTEM OVERVIEW

Background: Military borders and bases are difficult to monitor strictly off limited manned patrols due to the vast amount of land that needs to be covered. The U.S-Mexico border is 1,954 miles long with 16,605 agents patrolling; yet, it has been unanimously agreed that our department is undermanned compared to the amount of people attempting to illegally cross. Security needs to be improved and it is clear manpower and physical patrols are not sufficient nor an efficient use of our numbers.

Goal: implement a network of towers and drone surveillance devices to monitor and recognize entities inside or near restricted border zones. The cameras will have real-

time object detection on both visible and thermal imaging. The towers will have a solar power architecture for minimal power upkeep and maintenance and airborne drones will replace patrol routes to free up manpower. The network coverage will allow manpower to be reallocated on a rapid-deployment basis when entities are encountered. The real-time entity information allows properly sized manpower to be sent to handle potential threats, leading to an efficient use of available resources while increasing border safety.

1.4 MAIN FUNCTIONS

The major functions that the PMB performs can be seen in Figure 1-4 and are described as follows:

- Detecting Objects: Using thermal imaging with edge computing and intelligent algorithms, sensor and imaging hardware are used to detect and report physical changes within a specific area. Clear differentiation of actual person and other objects (i.e. vehicle, animal, etc.).
- Object Recognition: The object recognition task is the process of utilizing model-based techniques to identify potential threats. Several approaches can be applied to classify new detected objects.
- Tracking Objects: Tracking techniques can be split in two main approaches: 2D models with or without explicit shape models and 3D models.
- Behavioral Analysis: This stage broadly corresponds to a classification problem of the time-varying feature data that are provided by the preceding stages.
- Transmitting Data: Secure wireless data transmission.
- Mapping: Location identification using technologies like GPS and SLAM.
- Sending Alert: AI automatically detects changes, both expected and unexpected, and immediately communicates alarms if needed. It acts as a proactive autonomous guard that dramatically reduces false alarms.
- Recording/ Storing Data: Records any detected object and stores as positive, false positive, negative, or false negative.

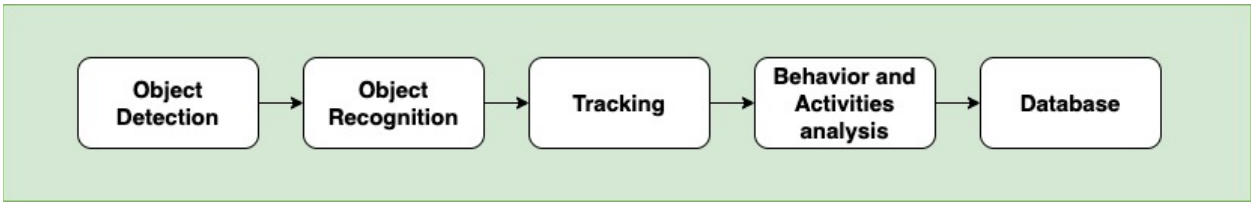

Figure 1-4 Traditional Flow of Processing in Visual Surveillance

1.5 CRITICAL USE CASE REQUIREMENTS

Figure 1-5 represents use case models that satisfy the PMB System requirements. The purpose is to provide a graphical notation of the requirements and their interaction within the whole system.

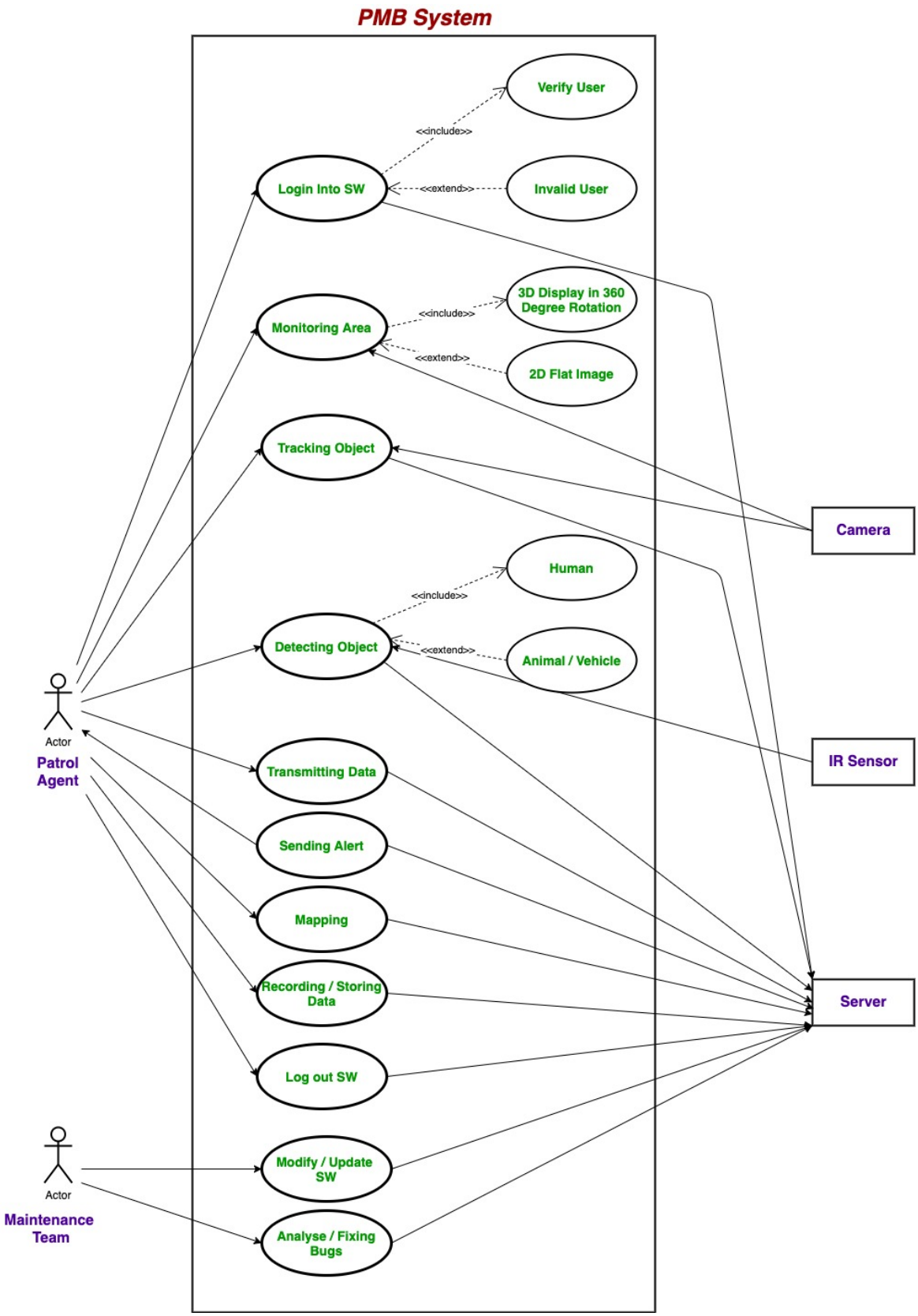

Figure 1-5 Use Case Diagram

1.6 ENVIRONMENT

The Lattice platform is a network, developed and used by the parent company Anduril, with an AI backbone that uses computer vision, machine learning and mesh networking to fuse together data from radar, optical, thermal and RF sensors to provide a unified picture of the surroundings, and can effectively classify targets (drone, bird, car) and automatically alert operators.

A.I. Software Package: Artificial intelligence program that takes transmitted detected objects as input and outputs an alert if it detects a positive match for a security risk.

SECTION 2. REFERENCED DOCUMENTS

The documents listed below were either used to create this SDP or are referenced in it.

2.1 GOVERNMENT SOURCES

MIL-STD-498 (United States Military-Standard-498): uniform requirements for software development and documentation.
https://en.wikipedia.org/wiki/MIL-STD-498

MIL- STD-1521 (United States Military-Standard-1521) Technical Reviews and Audits for Systems, Equipment, and Computer Software.
https://en.wikipedia.org/wiki/System_Design_Review

Software Development Plan, Data Item Description DI-IPSC-81427
https://apps.dtic.mil/dtic/tr/fulltext/u2/a291266.pdf

2.2 GENERAL SOURCES

- https://www.bizjournals.com/losangeles/news/2020/07/02/anduril-wins-cbp-deal-to-install-ai-surveillance.html
- https://algorithmia.com/blog/hardware-for-machine-learning
- https://en.wikipedia.org/wiki/Infrared_detector
- https://www.vox.com/recode/2019/5/16/18511583/smart-border-wall-drones-sensors-ai
- https://www.anduril.com/work
- https://www.wired.com/story/palmer-luckey-anduril-border-wall/
- https://www.washingtonpost.com/immigration/trump-virtual-border-wall/2020/07/02/7b380490-b0ac-11ea-a567-6172530208bd_story.html
- https://www.visualcapitalist.com/millions-lines-of-code/
- https://www.geeksforgeeks.org/software-engineering-cocomo-model/
- https://www.indeed.com/career/software-engineer/salaries/CA
- SW Dev Plan NASA for ECS cd3080108.pdf

SECTION 3. OVERVIEW OF REQUIRED WORK

Based on the *selected Process Model*, table 3-1 defines an overview of the PMB system. This strategy helps to evolve the main functional capabilities of the software.

Process Model	Well Defined All Requirements at First Phase	Multiple Software Development Life Cycle	Workflow
Waterfall	Yes	No	Linear
Incremental	Yes(Reasonably)	Yes	Linear & Parallel
Evolutionary	No	Yes	Spiral & Prototyping

Table 3-1 Key Characters of Three Main Process Models

The PMB system will be developed using the *Incremental Process Model*. All requirements will be well defined/understood before the project starts. The overall flow of development is linear. During each phase of development one increment will be delivered, beginning with a preliminary top-level design at first and more detailed design as development progresses, until the final product is produced. We will fundamentally complete the software development process multiple times throughout the line of different increments. During each increment we will have specific goals and discrete systems being taken to completion. PMB's goal is designing software to increase border security as well as efficiency using Artificial Intelligence(AI).

The incremental design process will allow client's updated requirements to be considered for subsequent increments. This will allow us to receive feedback on recent increments of systems and maintain the overall health of the plan. It is illustrated in Figure 3-1.

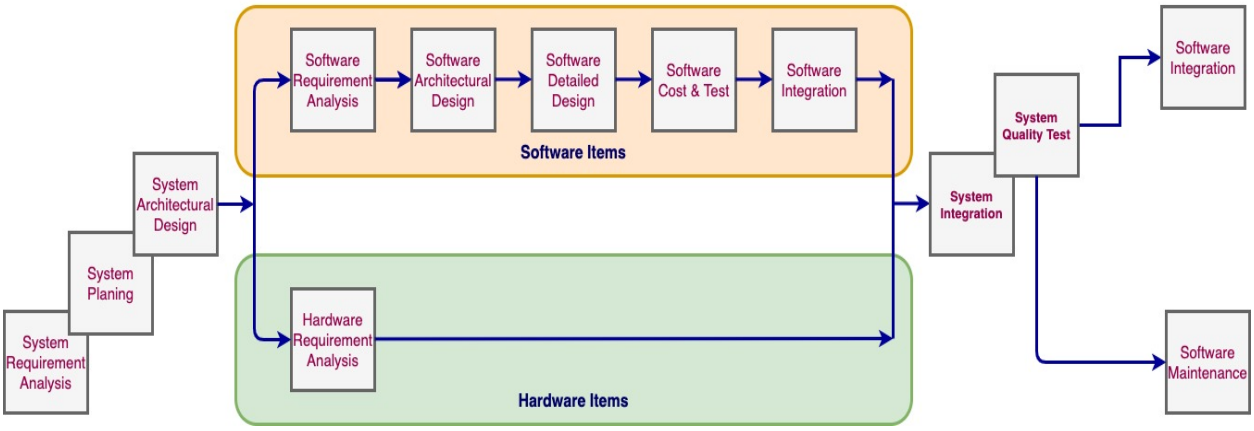

Figure 3-1 PMB Project Workflow

Supporting processes encompassing:

- Documentation
- Software Configuration Management(SCM)
- Software Quality Assurance(SQA)
- Verification & Validation
- Review & Audit

Organizational processes encompassing:

- Management
- Infrastructure
- Improvement
- Training

Software Development/Test Team organization and their responsibilities is illustrated in Figure 3-2.

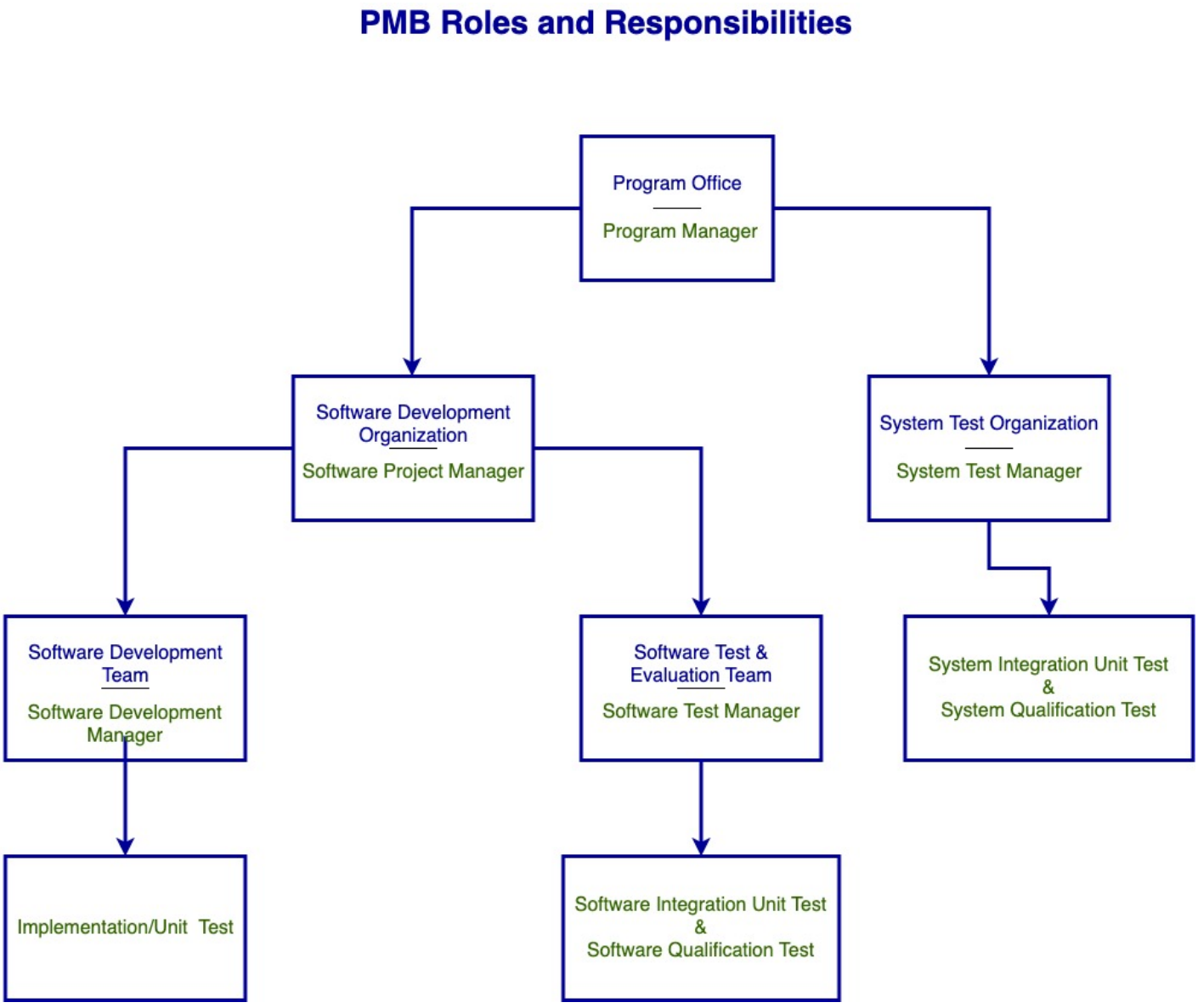

Figure 3-2 PMB Project Roles & Responsibilities

SECTION 4. PLANS FOR PERFORMING GENERAL SOFTWARE DEVELOPMENT ACTIVITIES

4.1 SOFTWARE DEVELOPMENT PROCESS

The MIL-STD 498 Incremental Life Cycle Model has been used to guide the content and format to develop the SDP for PMB software.

4.1.1 REQUIREMENTS

The first phase is started by Requirements analysis, putting together all user requirements through documentation, client interviews, observation, and questionnaires. These communication activities can happen in a meeting with software/hardware engineering and U.S. Customs and Border Protection to help define the overall goal of PMB software and the scope of the project for our clients. In accordance with the main objective of the project, several requirements may be discovered.

4.1.2 DESIGN

During the design phase we will design the architecture, modules interfaces and input/output data for the system utilizing the requirements from the relevant SRS document. Our first step is to take the requirements and build an initial architecture prototype. We take this broad set of ideas and guidelines and organize it into functioning areas. Each of these areas are then put through the same process to break them up into smaller systems and subsequent software development tasks. Eventually we will have a blueprint for the larger system. This iterative process involves coming up with an idea, getting feedback, reworking it, and repeating the process until the system passes quality assurance.

The Design and Coding phase will be done by the Software Development Team using Object Oriented (OO) design methods.

4.1.3 UNIT CODE AND TEST

During this step a detailed description of what objectives the system should meet is defined, after which software developers can begin the coding process.

Unit testing is focused on the smallest unit in a piece of software. We are trying to isolate different areas, repeating until we test every module of the program. In this, we take a module, then we give it test cases with expected, unexpected, and bound values.

4.1.4 SOFTWARE INTEGRATION

Software integration involves integrating a software unit into it to the system it belongs to. Integration occurs before testing but must be planned before implementation. To get integration correct, we must prepare in advance which means it also fits into the implementation section. The level of planning here directly relates to how large the scope of the deployment is.

4.1.5 TESTING

We will test the completed integrated system in order to detect any possible defects within the system. The testing team is independent from the development team, they test the quality of the system impartially. These tests can be done in functional and non-functional categories.

4.1.6 SOFTWARE MAINTENANCE

Maintenance tasks should always be designed with the idea of retreat. If something goes wrong, we can change the initial software and update the whole system. During this phase we will develop, test and deploy the software as a modification version to make sure all operational functions satisfy the requirement of the product in a manner of time. The maintenance team must track the software problems during this phase. Any changes must be recorded in their corresponding documents.

We use an Activity Diagram to model both computational and organizational workflows of software processes, it shows the flow of control in a system, Figure 4-1 illustrates sequential and concurrent behavioral UML Activity Diagram.

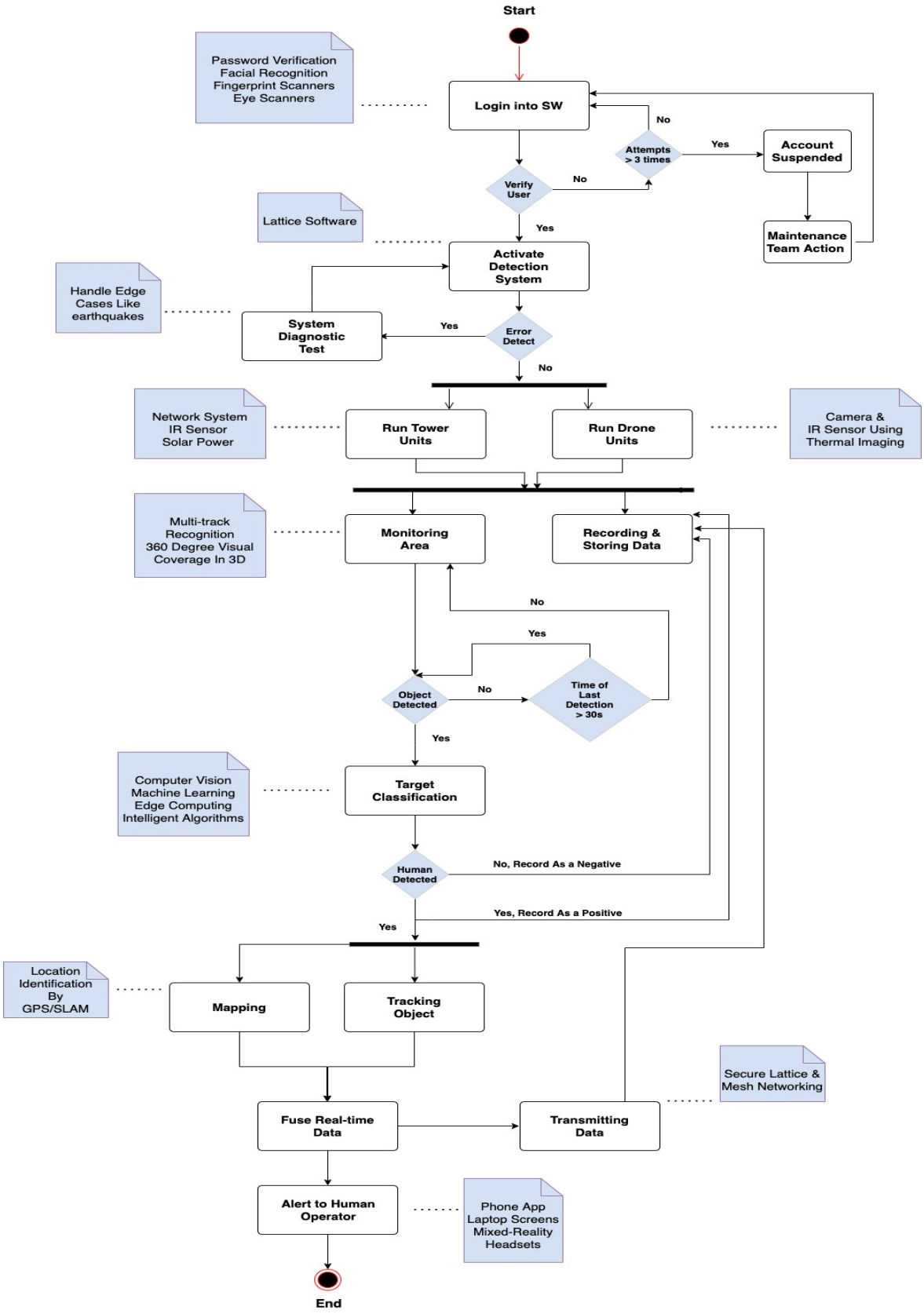

Figure 4-1 UML Activity Diagram

4.2 STANDARDS FOR SOFTWARE PRODUCT

PMB software is developed according to the documents listed in figure 4-2.
These documents apply standards that are appropriate to software requirements, design, coding, testing, and data.

Document	Description
MIL-STD-498	Software Development and Documentation
MIL-STD-961D	DoD Standard practice Defense Specification
MIL-STD-973	Configuration management
MIL-STD-1521	Technical Reviews and Audit for System

Table 4-2 PMB Project Standard

4.3 HANDLING OF CRITICAL REQUIREMENTS

4.3.1 SAFETY ASSURANCE

The software developers are responsible for identification of safety-critical failures that can lead to an unsafe system state. The goal is to evolve a safety assurance procedure, including both tests and analyses, to satisfy the requirements, design, implementation, and operating procedures to minimize or eliminate the potential for hazardous conditions.

4.3.2 SECURITY ASSURANCE

Software developers, integrators, and testers are responsible to identify security-critical failures in order to make sure the PMB system meets the security requirements. It can be done by evaluation activities like checking the security manual standards, then security assurance procedures will be reviewed by the security office.

4.3.3 PRIVACY ASSURANCE

Privacy requirements should have been factored into design and development from the early stages of planning. One effective method to evaluate privacy assurance consists of two different points of view:

1. Privacy must be acknowledged in the organization.
2. Appropriate privacy policies should be clearly defined.

4.4 COMPUTER HARDWARE RESOURCE UTILIZATION

Computer resource utilization is characterized in two different categories:

- Development Environment - Each part of computer hardware resource utilization has a detailed schedule that will be established by the Software Project Manager (SPM). This schedule identifies anticipated users, purposes, and scheduled time to support analysis, software design, coding, integration, testing, and documentation. It can resolve conflicts in sharing resources used by multiple users. The SPM will manage resource needs with development, integration, and test groups.
- Operational Users Environment - The SPM will establish and maintain a database of the site-specific computer hardware and commercial software resources. It will address resources by hardware configurations and commercial software licensing requirements. If a specific site’s computer hardware resource or licensing needs are insufficient for a planned build, then those needs will be

communicated to those sites notifying them of the configuration enhancements needed for the next build.

4.5 RECORDING OF RATIONALE

All main decisions in technical and in organizational processes will be recorded in a software engineering notebook (SEN). This notebook includes all key issues documented to support any result and conclusion. Additional rationale for SDP will be documented in the updated version of software development plan.

SECTION 5. PLANS FOR DETAILED SOFTWARE DEVELOPMENT ACTIVITIES

5.1 PROJECT PLANNING AND OVERSIGHT

5.1.1 SOFTWARE DEVELOPMENT PLANNING

The structure of the software development team shall consist of a senior software architect, software engineer team leader, and software engineers. Additionally, there shall be a quality assurance engineer. The architect shall delegate software tasks for the entire project to team leaders who then break these tasks into discrete scheduled unit development assignments for the software engineers. Following Figure 5-1, the software project planning and oversight process.

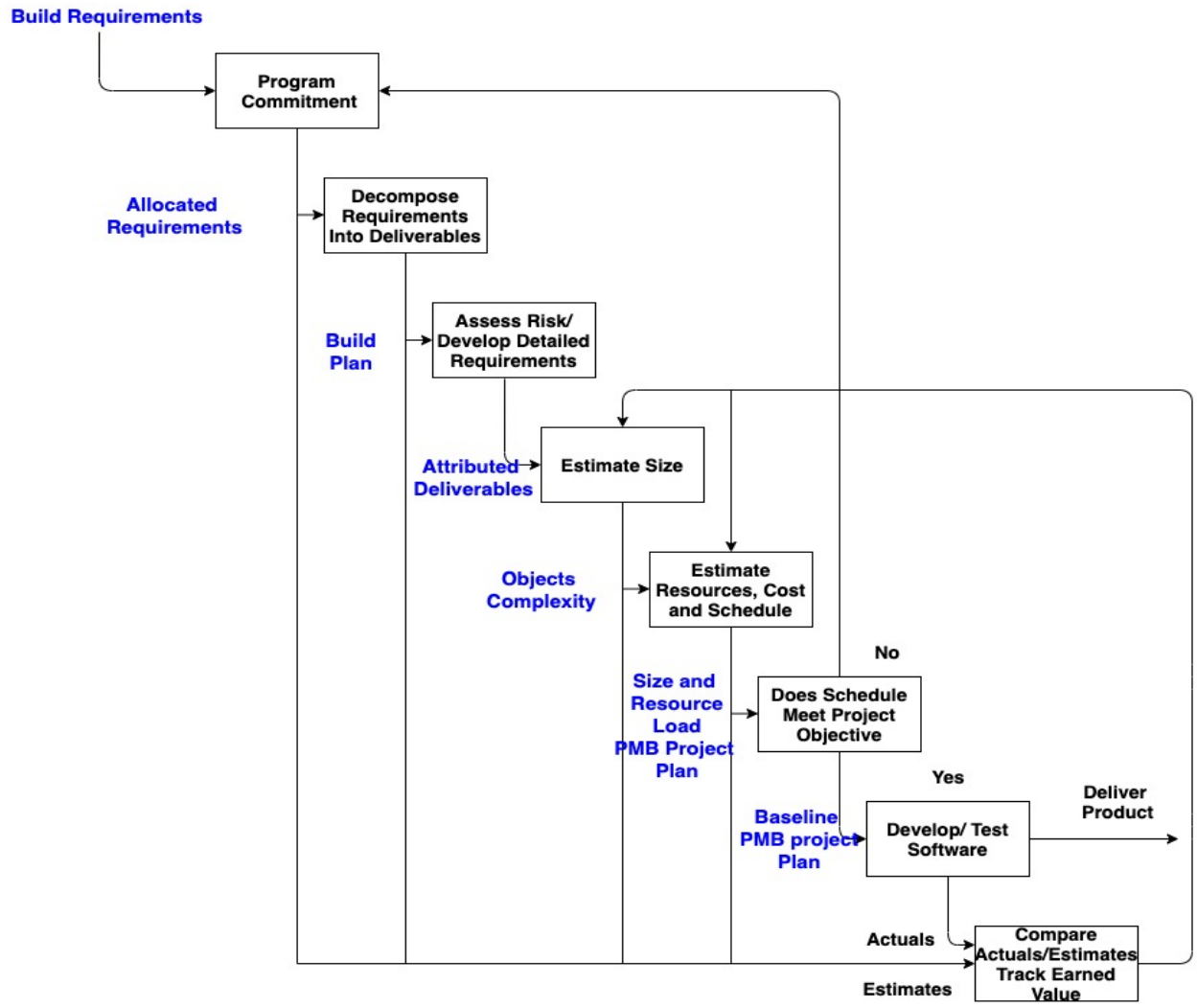

Figure 5-1 Software Project Planning & Oversight Process

5.1.2 TEST PLANNING

Team leaders shall be responsible for unit code tests that ensure all requirements as listed in the SRS are met for delegated tasks.

5.1.3 SYSTEM TEST PLANNING

System Test Planning Code implementations shall be tested utilizing virtual machines simulating deployed hardware conditions by software engineers who completed the implementation.

5.1.4 SOFTWARE INSTALLATION PLANNING

An embedded systems engineer shall oversee software installation to test hardware with unit code tests that ensure all functional and non-functional requirements as listed in the SRS are met.

5.1.5 SOFTWARE TRANSITION PLANNING

Following and Updating Plans, including Intervals for Management Review. At each stage of software development, a software quality assurance team member shall verify that the task meets all design and testing requirements for review by the senior software architect. (or other managerial position)

5.2 ESTABLISHING A SOFTWARE DEVELOPMENT ENVIRONMENT

The PMB Development System (PDS) will be maintained on a secure, UNIX-based system with encrypted endpoints for added security due to the nature of the project. The code will be backed up daily. The PDS environment is a compilation of several hardware and software tools that act in combination to support the software development process and will be referenced as a whole for readability; however, should be interpreted contextually for its specific parts.

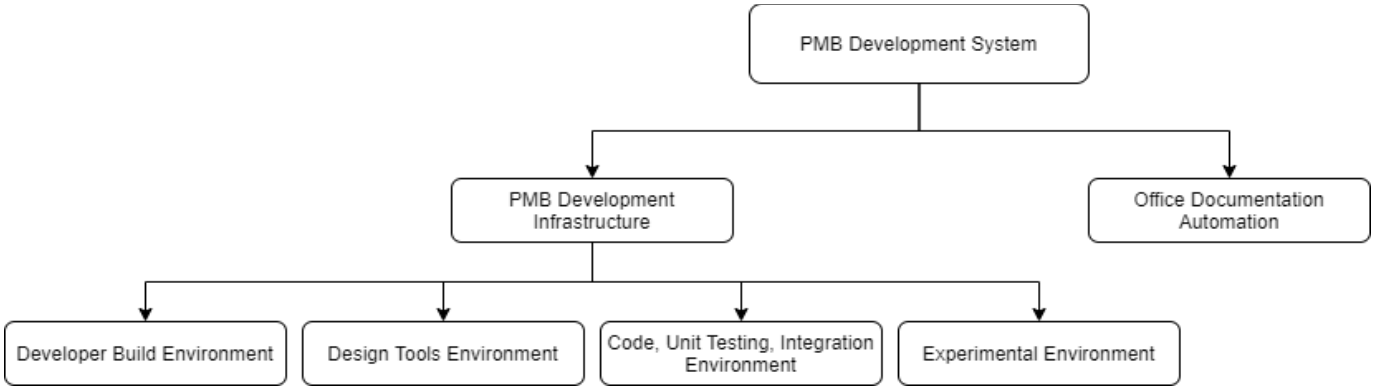

Figure 5-2 PMB Development System Environments

The PMB Development Infrastructure logically separates the environments used for specific aspects of the software production cycle; but this is not to say the environments are separate machines necessarily or that they are not able to run concurrently from one another.

Environment	**Description**	**Typical Users**
Developer Build Environment	Computers building the software during code/ unit tests. Established to baseline the platform PMB will be running on for deployment.	Developers
Experimental Environment	Computers used for evaluating, prototyping and demonstrating the software version.	Developers/Project Leads
Code, Unit Testing, Integration Environment	Computers used for changing, unit testing, and integrating the changes if testing passed.	Developers
Design Tools Environment	Computer supporting CASE tools for design relevant model renderings during release.	Project Leads
Office Automation Environment	Computers used to create documentation, presentations, and other aspects of the project that do not involve the code directly.	All PMB Personnel

Table 5-2 PMB Environment Description

5.2.1 SOFTWARE DEVELOPMENT ENVIRONMENT EVALUATION

As plans for our project involve scaling and evolving, our environment will need to be able to do the same. To meet this end, establishment of a support and maintenance team for the PDS is necessary. Additionally, a PDS Change Control Board will be responsible for determining what software upgrades and configuration changes will be included in future PDS environments. All implemented changes will be documented and tracked for future maintenance purposes.

5.3 SYSTEM REQUIREMENTS DEFINITION

During the development process, this phase will be performed during every increment to define the functionality and environment for the whole system. There are two parts to the system requirements definition phase: System Requirements Analysis and Software Requirements Analysis. The System Requirements Definition (SRD) will have responsibility for these activities.

5.3.1 SYSTEM REQUIREMENTS ANALYSIS

System requirements will encompass both hardware configuration and software configuration items. For hardware, documentation delineating tower, drone, and computer machinery will be necessary while the software configuration items will have overlap between systems. The functional and performance requirement documents will provide details on system level requirements for PMB and it will be the duty of the SRD to ensure allocation of these specifications are met prior to progressing through development.

As development continues, a high-level description of the system's capability will be documented for tracking purposes.

5.3.2 SOFTWARE REQUIREMENTS ANALYSIS

Software necessary for system functionality will be defined and an approach to solving it will be determined. In addition to developing needed software configuration items, potential candidates for software reuse (heritage software

or commercial-off-the-shelf (COTS) software) may be explored during the analysis. The conclusion of the software requirements analysis is marked by a peer review in which system capabilities are reviewed. The Software Program Manager (SPM) and the Program Manager (PM) will be participants in this review to ensure that operations development have met requirements. After any defects have been identified and corrected, a revision to the traceability matrix will be made and subject to SPM or PM approval.

5.4 SOFTWARE DESIGN

All software requirements will be analyzed and modeled. The software elements and information from requirement analysis will allow us to create the design model. For PMB software Object Oriented (OO) is a more appropriate approach than procedural. Therefore, all definitions and notations should follow OO methodology.

This methodology helps:

- In giving a modular approach to analysis and design to transform the information created during requirement analysis into the data structure that is necessary to move the project into the implementation phase.
- Another reason for using OO is because experts in video-based surveillance are mainly familiar with OO technology.
- OO techniques reduce complexity within design and are suitable for prototyping.
- OO implementations are flexible and easily accommodate changes so reducing maintenance costs, and that OO techniques can provide other important benefits like extensibility and reusability.

Design and analysis results can be stored in libraries that can be used again at a later time.

Software design is an incremental process. Top-level design for the PMB project is part of architectural design to identify the relationships between all major structures and architectural patterns. During detailed design, necessary capabilities for the iterative process are established that should meet an appropriate level of sufficiency for implementation. All the updates and refinements to the model has an incremental

process during detailed design. A software architectural model of PMB Project will be placed under developmental configuration management following the top-level design review (TLDR).

5.4.1 ARCHITECTURAL DESIGN

A high-level design will be generated for each discrete system. These design documents will then be used to guide the generation of Program Design Language (PDL) documents in the detailed design phase.

5.4.2 DETAILED DESIGN

The Detailed Design phase begins with generating an up-to-date list of detailed design elements to be prescribed in a PDL which will be used to detail reproducible algorithms used in each code unit. Additionally, code objects will have all attributes, data types, and all methods defined in the PDL.

5.5 SOFTWARE IMPLEMENTATION

This phase can be broken down into the following sections

5.5.1 SOFTWARE CODING

Each module of software will be coded with a clean compilation. Proper naming standards and documentation procedures will be followed as outlined by the standards in section 4.2. After each module is completed, a step-by-step unit test procedure will be developed to verify requirements have been met. Once verified, the module can be fully integrated into the system.

5.5.2 PEER REVIEW

Peer reviews will be conducted to ensure code compiles properly while meeting allocated requirement specifications. Besides code and design artifacts, unit tests will also be subject to peer review.

5.5.3 UNIT TESTING

After peer review and satisfying any defects observed, the software module will undergo unit testing to validate requirement satisfactions. Details on the process will be defined in section 5.6.

5.5.4 MERGING

After successful unit testing and validation procedures, the module can be merged with the baseline. This process will be done as a change request with appropriate documentation that supports requirements have been met and a program manager will sign for it. Once the merge has been completed, the implementation phase will conclude and move onto integration.

5.6 UNIT INTEGRATION AND TESTING

Purpose: To ensure all requirements are met and every unit is operational such that there is no catastrophic failure of hardware or software resulting in missed detection of a threat from the PMD.

Roles and Responsibilities: The producer of the unit for testing will conduct a unit test to verify the unit meets all requirements. The results will be recorded and signed off by the software developer's team leader.

Each unit is coded based on the design approved by the PM along with a specific program language that has been chosen based on some requirement related criteria (i.e. space efficiency, real-time processing, etc.). The modules will be debugged to the point of successful compilation to remove any syntax errors as entry criteria for the unit test.

Next, pass/fail criteria, measurable parameters, and tolerance will be specified and tested for designated test cases. All resulting output will be documented for configuration management purposes as exit criteria for the unit tests. If all tests pass, the module may be approved for merging into system baseline code. If there are any

issues, they shall be documented, marked for debugging, and scheduled for reassessment.

5.7 INTEGRATION AND TESTING

5.7.1 Preparing for Integration and Testing

Before integration can begin, the system must have gone through the Unit Integration and Testing phase and passed relevant tests.

5.7.2 Performing Integration and Testing

During this phase, the system must be checked to ensure proper functionality outside of the development environment.

5.7.3 Revision and Retesting

Depending on the severity of code compliance with specifications, the Revision and Retesting necessary for each iteration of a code fix will follow a formal review which will give the tester time to integrate code fixes.

5.7.4 Analyzing and Recording CSCI/HWCI Integration and Test Results

A Test Readiness Review (TRR) will be conducted at the end of the Revision and Retesting phase of Integration and Testing. The TRR will assess the readiness of the system for Qualification Testing. During this phase, the problems discovered during the Performing Integration and Testing phase will be prioritized and if found of sufficient risk, the system will be rejected for Acceptance Integration. If accepted, the system will be tested by a Quality Assurance Engineer whose task is to formally run tests on the system and generate data samples to be analyzed. A Consent to Ship document will then be created and a review will be held to finalize the system.

5.8 QUALIFICATION TESTING

Purpose: Qualification testing will be used to ensure the software integrated with hardware or into the base system meets all functional requirements.

Implementation of qualification testing will follow the same model outlined for unit testing in section 5.6. Once the software is integrated, it must pass the allocated requirements and procedures defined in the unit tests. All testing will be recorded and documented for configuration management purposes. Bidirectional traceability will be established between test cases to support coverage and consistency. If any deficiencies are found, a report is to be made to the Software Program Manager with summary of issue(s).

5.9 SYSTEM QUALIFICATION TESTING

Purpose: the overall system and subsystems must be verified to have met the necessary requirements outlined within this document. If there are multiple builds used throughout development, system qualification testing will be postponed until a final build is reached.

The System Testing Organization will be responsible for the system qualification testing on the final build. Testing will include an overall system requirements verification as well as independent subsystem verifications for accurate coverage. The testing shall be done on the targeted computer system chosen for client use so there will be no overlap upon release.

The first set of testing shall be documented with measurable parameters and result explanation as this will be included in product documentation to the client. If any deficiencies are found during testing, a revision report must be created with a determined reassessment date. The Program Manager will be responsible for approving the deficiency has been corrected.

5.10 SOFTWARE QUALITY ASSURANCE

5.10.1 SOFTWARE QUALITY ASSURANCE OVERVIEW

A Quality Assurance organization will be used to ensure the software developed, or procured through COTS, comply with the contractual requirements to promote the highest quality software standard. Quality of all code, excluding prototype code for testing, will be the joint responsibility of all PMB employees during development and maintenance.

5.10.2 QUALITY ASSURANCE ORGANIZATION

The PMB project approach to ensuring that all software meets the performance assurance requirements within this document is managed in two ways. First, the process and procedures for software development are documented herein and in the project, instructions referenced throughout this document. Second, the Quality Assurance team provides an independent monitoring, auditing, and corrective action function, which ensures that the approved software development process and procedures have been followed or a variance request has been approved. The PMB project development team will use reviews and testing procedures to uphold high software quality standards. The Quality Assurance organization will ensure the implementation of the approved software development processes and standards identified in this document. To accomplish this, the Quality Assurance Manager maintains a reporting role to the Program Manager (PM) and to the Software Program Manager (SPM). Any issues found by the QA organization can be quickly handled by open communication channels. Members of the Quality Assurance team will be assigned to monitor, audit and ensure corrective action of specific functional areas of the PMB project while remaining separate from the Software Test Organization (STO).

5.10.3 ROLES AND RESPONSIBILITIES

All members of the QA team will have an active role in Software Quality Activities (SQA), and their roles are defined here

- Quality Assurance Manager:

    - Ensures proper staffing of QA team members
    - Manage within QA budget
    - Conduct performance evaluations of QA staff
    - Coordinate audits for PMB development
- Quality Assurance Engineers:
    - Ensure appropriate standards are selected, implemented, and followed strictly by performing evaluations against consistent criteria
    - Promote problem avoidance by prompt evaluation dissemination with recommended corrective action
- Quality Assurance Team Members:
    - Identify and report any inability to perform procedure to QA Manager or QA Engineer through variance request
    - Provide information to conduct quality evaluation
    - Identify and recommend solution to discrepancy issues

5.10.4 REQUIREMENT FULFILMENT

MEMORY CAPACITY

- Control of memory utilization will be based on common storage and function utilization as well as efficient coding. The memory amount being utilized will be kept track by the REA throughout the coding development process. During code walk-throughs, it will provide a point at which allocated versus used memory will be critiqued. The use of memory will be kept track of and a complete memory map will be established to verify the usage of the memory in all states of software execution during the integration process.
- By using these control procedures, it will be possible to form and verify memory usage and to assure the customer that at least 5 percent spare memory capacity exists in the delivered system.

CPU EXECUTION TIME CAPACITY

- Control of the programmer utilization of the available CPU time in the Lattice Platform will be based on efficient coding as reviewed and

managed by the REA during the code walk-through. As the code development progresses, the execution time required by the software will be monitored often. During testing and integration, the execution time of the Lattice Platform will be measured to verify the time required for execution in all possible system modes.

- Laboratory verification of running time will be performed with the development station, the INTEL In-Circuit Emulator system (ICE) and its tools. The REA will monitor the throughput capacity and maintain records in the Software Development Notebook (SDN) for complete Data Processor CPCI's throughput usage.

5.11 SOFTWARE QUALITY ACTIVITIES

The Quality Assurance Engineer will perform quality assurance evaluations to provide insight into the use and adherence to the processes documented in the software development plan. There are two types of evaluations performed, audits and product evaluations. An audit is an objective examination of documented processes to verify that company and or contractual requirements are being met. A product evaluation is an objective examination of deliverable, non-deliverable, or non-developmental products to verify they are in agreement with company and /or contractual requirements. In addition to conducting quality assurance audits and product reviews, the Quality Assurance Engineer is responsible for creating/updating audit and product evaluation criteria, preparing and maintaining the results of audits, product evaluations, and cited deficiencies. The Quality Assurance Engineer will also provide periodic quality assurance status reports.

5.11.1 QUALITY ASSURANCE AUDITS

Quality Assurance Engineers shall evaluate the following processes:

- Architectural Design
- Detailed Design
- Implementation

- Integration
- Review

5.11.2 PRODUCT EVALUATIONS

QA Engineers shall conduct product evaluations in accordance with ITS Quality Assurance Procedures. These evaluations shall include:

- Database Specification
- Tower Implementation
- Drone Implementation
- Full Network Deployment

5.11.3 QUALITY ASSURANCE AUDIT CRITERIA

The QA Engineer shall tailor criteria for both audit and product evaluations to the task descriptions outlined to corresponding project and work instructions. All procedures must remain consistent and objective; additional criteria may be added with QA Manager approval.

5.11.4 QUALITY ASSURANCE REPORTING

All results from audits and product evaluations shall be documented and maintained by the QA Engineer. If deficiencies are present, they must be cited with a report of corrective action. Additionally, the QA Engineer may report a preventative action report (PAR) for QA Manager approval in an effort to minimize similar future deficiencies.

Status reports will also be prepared by the QA Engineer on a monthly basis to be handed to the Program Manager. The report shall address activities performed in the last reporting period as well as activities that are still currently in progress. Any open and recently resolved issues shall also be included in the report.

5.12 TECHNICAL AND MANAGEMENT REVIEWS

Team review should be formed of software engineers (technical reviewer) and others (leader reviewers). Each member of the review team has a set of specific tasks based on advanced planning, prepared agenda and meeting structure. All the results should

be formally recorded as a review issue list to follow up by the designer group in order to resolve at a later time. During the review process defining appropriate levels of detailed reports can cause immediate errors discovery that might otherwise propagate further into the other software processes.

5.12.1 TECHNICAL REVIEWS

Technical Review (TR) is part of software quality control activities as the most effective filter by narrowing the focus on a specific and reasonably small part of overall software. Depending on the work product, timeline and the team members, specific levels of formality should be considered as part of review tasks.

Main objective of TR:

- Focus on a work product to uncover potential errors in logical processes and implementation of software.
- Verify software is demonstrated by appropriate technical solutions to resolve the technical issues.
- Software satisfied by predefined standards and developed in a uniform manner.
- Make the project more manageable by ensuring ongoing communication between review team and leader team.

5.12.2 MANAGEMENT REVIEWS

Software project managers should plan and participate in management reviews to resolve issues that remain from the technical reviews. It is an effective way to identify management-level issues and risks that are not discovered in TR. The review meeting should be included by the people with an authority to make cost and schedule decisions. It helps to keep the management team informed about the project status. The review should be scheduled in project plan and can be in any combination of these reviews:

- Software plan reviews

  Software development plan (SDP)

  Software test plan (STP)

- Software requirement reviews
- Software design reviews

    Software design decisions

    Architectural design

    Detailed design

- Test readiness/result reviews
- Software usability/supportability reviews
- Critical requirement reviews

5.13 SOFTWARE CONFIGURATION MANAGEMENT

Software Configuration Management will be performed under the direction of the SCM Manager according to the processes and procedures defined in the PMB Project Software Configuration Management Plan. It helps to check the consistency of the product process where each build can be refinements of previous builds.

5.14 OTHER SOFTWARE DEVELOPMENT ACTIVITIES

5.14.1 RISK MANAGEMENT

In the context of the risk area, identification and management of risk are critical for the success or failure of a software project. One effective strategy to handle the risk concerns of a project is to identify the potential risk, their probability, impact and consequences before starting technical work. Then the software team establishes a contingency plan to deal with management of risks.
Risk management is an iterative process and the Risk Information sheet(RIS) should be updated and visited regularly. Risk table illustrates risk estimation in a simple way Table 5-3 & 5-4.

Severity	Likelihood				
	Rare 1	Unlikely 2	Possible 3	Likely 4	Almost certain 5
Catastrophic 5	5	10	15	20	25
Major 4	4	8	12	16	20
Moderate 3	3	6	9	12	15
Minor 2	2	4	6	8	10
Negligible 1	1	2	3	4	5

Table 5-3 Risk Probability/Severity Distribution Table

Rare 1-20 %
Unlikely 21-40 %
Possible 41-60 %
Likely 61-80 %
Almost certain 81-100%

Risk Rating = likelihood * Severity

Stop or Urgent Action

Action

Monitor

No Action

Risk Table

Risks	Category	Probability	Impact
Incorrect Time Estimation	Project Risk	60%	12
Improper Resource Allocation	Project Risk	30%	8
Unexpected Expansion of Project Scope	Project Risk	50%	12
Improper Plan to Handle Leading-Edge Technology	Technical Risk	70%	16
Improper Budget Estimation	Project Risk	90%	25
Cost Overruns Due to Underutilization of Resources	Project Risk	80%	12
Improper Tracking of Finance	Project Risk	60%	15
Conflicting Priorities	Technical Risk	80%	8
Improper Process Implementation	Technical Risk	30%	10
Breakdown in Communication	Project Risk	50%	9
Improper Maintenance Plan	Technical Risk	80%	16
Lack of Sufficient Training on Tools	Project Risk	80%	8
Improper Design Strategy	Technical Risk	50%	15
Project Plan Not Meeting Requirements	Technical Risk	70%	16
End Users Resist System	Business Risk	40%	4
Interface Design Not Meeting Expectation	Technical Risk	50%	12
Delivery deadline will be tightened	Business Risk	50%	12
Changes in government policy	Business Risk	20%	3

Table 5-4 Risk Table

SECTION 6. PROJECT ESTIMATION

6.1 COCOMO MODEL

Constructive Cost Model (COCOMO) is based on Lines of Code (LOC) where project estimation is done based on the total lines of codes required to develop the system. I.e. Size of the system defines the cost of the project. This model was created by Barry W. Boehm in 1981 and is used to estimate the effort, cost, development time, average staff size, productivity, etc.

The COCOMO Model consists of a hierarchy of three increasingly detailed and accurate forms. Here are the different types of COCOMO model:

1. Basic COCOMO Model - Used for quick and slightly rough calculations of Software Costs.
2. Intermediate COCOMO Model - Takes these Cost Drivers into account.
3. Detailed COCOMO Model - Accounts for the influence of individual project phases, i.e. in case of Detailed it accounts for both these Cost Drivers, and also calculations are performed phase-wise henceforth producing a more accurate result.

We will be using the Basic COCOMO Model for quick and rough estimates for the Lattice platform. The Basic COCOMO Model is broken into three different (organic, semi-detached, and embedded) categories. Boehm's definition of organic, semi-detached, and embedded systems is as follow:

1. Organic - A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. Semi-detached - A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environments lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience

and better guidance and creativity. For instance, compilers or different Embedded Systems can be considered of the Semi-Detached type.

3. Embedded - A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Basic COCOMO Model formula:
Where a, b, c, and d are constant values given in the table below.

Effort = $a(KLOC)^b$, where the unit is Person-Month
Development Time = $c(Effort)^d$, where the unit is Months
Average Staff Size = Effort / Development Time, where the unit is Persons
Productivity = KLOC / Effort, where the unit is KLOC/Person - Month

Type	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 6-1 COCOMO Constants

6.1.1 BASIC COCOMO MODEL LATTICE PLATFORM CALCULATION

The control software to run a U.S. military drone uses 3.5 million lines of code. A Boeing 787 uses 6.5 million lines of code to run its avionics and online support system. Given these two estimations, we estimate the Lattice Platform to have roughly 5 million lines of code.

KLOC = Kilo Lines of Code
We are given LOC = 5,000,000
Therefore, KLOC = 500

1. Organic: KLOC = 500, a = 2.4, b = 1.05, c = 2.5, d = 0.38

   Effort $= a(KLOC)^b$, where units is Person-Month
   $= 2.4(500)^{1.05}$ Person-Months
   = 1637.31 Person-Months
   = 1638 Person-Months

   Development Time $= c(Effort)^d$, where units is Months
   $= 2.5(1638)^{0.38}$ Months
   = 41.63 Months
   = 42 Months

   Average Staff Size = Effort / Development Time, where the unit is Persons
   = 1638 Person-Months / 42 Months
   = 39 Persons

   Productivity = KLOC / Effort, where the unit is KLOC/Person - Month
   = 500 / 1638 Person-Months
   = 0.31 KLOC/(Person - Month)

2. Semi-detached: KLOC = 500, a = 3.0, b = 1.12, c = 2.5, d = 0.35

   Effort $= a(KLOC)^b$, where units is Person-Months
   $= 3.0(500)^{1.12}$ Person-Months
   = 3162 Person-Months

   Development Time $= c(Effort)^d$, where units is Months
   $= 2.5(3162)^{0.35}$ Months
   = 41.97 Months
   = 42 Months

Average Staff Size = Effort / Development Time, where the unit is Persons

= 3161 Person-Months / 42 Months
= 75.29 Persons
= 76 Persons

Productivity = KLOC / Effort, where the unit is KLOC/Person - Month

= 500 / 3162 Person-Months
= 0.16 KLOC/(Person - Month)

3. Embedded: KLOC = 500, a = 3.6, b = 1.20, c = 2.5, d = 0.32

Effort $= a(KLOC)^b$, where units is Person-Months
= $3.6(500)^{1.20}$ Person-Months
= 6238.3 Person-Months
= 6239 Person-Months

Development Time $= c(Effort)^d$, where units is Months
= $2.5(6239^{0.32}$ Months
= 40.96 Months
= 41 Months

Average Staff Size = Effort / Development Time, where the unit is Persons

= 6239 Person-Months / 41 Months
= 152 Persons

Productivity = KLOC / Effort, where the unit is KLOC/Person - Month

= 500 / 6239 Person-Months

= 0.08 KLOC/(Person-Months)

The Lattice Platform will fall under the embedded category since the Lattice Platform is managing drones, towers, and other hardware needed to connect to the platform. This requires the highest level of complexity, creativity, and experience requirements. The software will require a large team size.

6.1.2 ESTIMATION SUMMARY

Time Allocation: 41 Months

People Power Requirements: Average Staff Size is 152 people

Cost:
The average monthly base salary according to indeed.com as of Oct 31,2020 is $9,308 for a Software Engineer in California.

From the COCOMO Model, we take the average staff size and time allocated for this project and multiple them together with the average monthly salary.

152 * 41 * $9,308 = $58,007,456 USD

6.2 OTHER FUNCTIONAL ESTIMATION REQUIREMENTS

6.2.1 QUANTITY

Sentry Tower – 32 feet towers will cover 2-mile radius
Ghost 4 sUAS - Autonomous VTOL sUAS, modular, man-portable, waterproof, and combines long endurance.
“A concrete structure 30 feet high that takes four hours to penetrate costs $24.5 million a mile,” he says. “A smart wall, a system like what Auduril is proposing, is about a half million a mile.” - Hurd, former CIA agent, a member of Congress with a computer science degree.
The United States-Mexico border is 1,954 miles long. Given these statistics, to protect the United States-Mexico border, the CBP would need about 977 Sentry

Towers (Autonomous Surveillance Towers) and a few hundred Ghost 4 sUAS drones for direct reporting. The CBP would also need the Lattice AI software to synchronize the hardware.

6.2.2 COVERAGE

The United States-Mexico border is 1,954 miles long. In addition to the maritime boundaries of 18 miles in the Pacific Ocean and 12 miles in the Gulf of Mexico. The infrastructure of the PMB must be able to reach at least 1,984 miles. The coverage of the infrastructure should account for depth of the border, which should be no less than 6 feet according to The Department of Homeland Security. This means the total area of the infrastructure should cover at the minimum 62,853,120 square feet (11,904 square mile).

SECTION 7. PROJECT SCHEDULES AND ACTIVITY NETWORK

Development schedule will follow figure 7.1

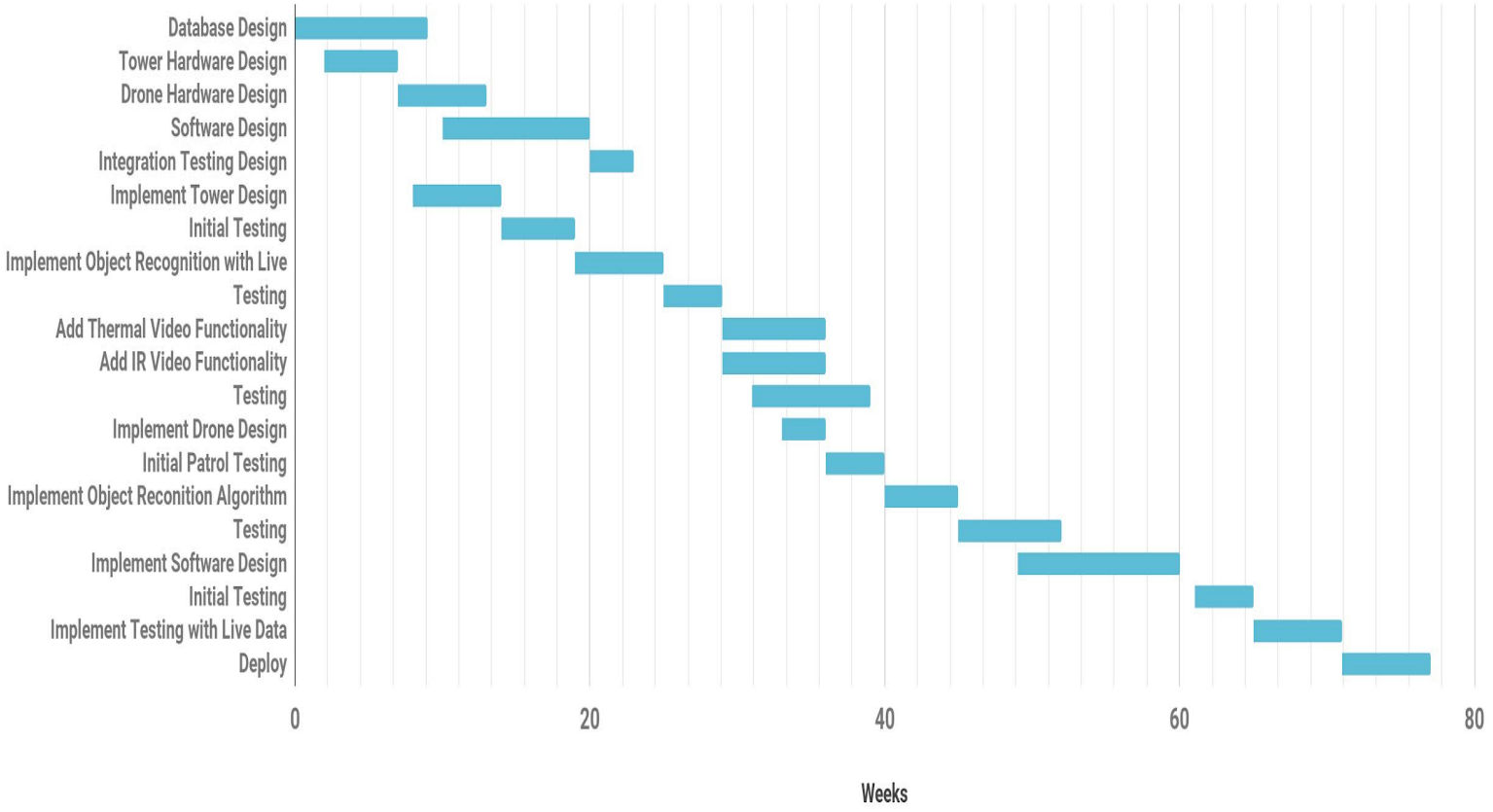

Figure 7-1 PMB Gantt Chart

SECTION 8. NOTES

8.1 TERMINOLOGY

AI	- Artificial Intelligence
CBP	- Customs & Border Protection
CM	- Configuration Management
CMO	- Configuration Management Office
COCOMO	- Constructive Cost Model
CSCI	- Computer Software Configuration Item
DID	- Data Item Description
GPS	- Global Positioning System
IR	- Infrared
KLOC	- Kilo Lines of Code
LOC	- Lines of Code
OO	- Object Oriented
PAR	- Preventative Action Report
PDL	- Program Design Language
PDS	- PMB Development System
PMB	- Protect Military Bases
PM	- Project Manager
REA	- Responsible Engineering Activity (team leader)
RIS	- Risk Information Sheet
SCM	- Software Configuration Management
SDLC	- Software Development Life Cycle
SDP	- Software Development Plan
SEN	- Software Engineering Notebook
SEN	- Software Engineer Notebook
SLAM	- Simultaneous Localization and Mapping
SPM	- Software Program Manager
SQA	- Software Quality Assurance
SRD	- System Requirements Definition
STO	- Software Test Organization
TLDR	- Top-Level Design Review

TRR - Test Readiness Review
TR - Technical Review