SAGRAD SOFTWARE DEVELOPMENT PROCESS

September 22, 2023

Sagrad Document #: SG 906-0002

Revision: A2.0

Status: Draft

Sagrad, Inc.

Corporate Headquarters

202 West Drive

Melbourne, Fl 32904

Telephone: (321).726-9400

Sagrad.com

1 COPYRIGHT, TRADEMARKS, DISTRIBUTION NOTICES

1.1 COPYRIGHT

Information in this document may change without notice and does not represent a commitment on the part of Sagrad®.

Sagrad® claims copyright in this documentation as an unpublished work, revisions of which were first licensed on the date indicated in the preceding notice. The claim of copyright does not imply a waiver of Sagrad's other rights. See Notice of Proprietary Rights.

1.2 TRADEMARKS

The Sagrad® logos are registered trademarks of Sagrad®.

All other company and product names used herein may be the trademarks or registered trademarks of their respective companies.

1.3 NOTICE OF CONTROLLED UNCLASSIFIED INFORMATION

This documentation is a trade secret and the property of Sagrad® and may contain Controlled, Unclassified Information (CUI). Unauthorized access, use, disclosure, or distribution of this information, in whole or in part, without proper authorization, is prohibited by law.

1.4 DESTRUCTION NOTICE

For classified documents, follow the procedures in DOD. 5220.22-M, National Industrial Security Program Operating Manual, February 2006, Incorporating Change 1, March 28, 2013, Chapter 5, Section 7, or DoDM 5200.01-Volume 3, DoD Information Security Program: Protection of Classified Information, Enclosure 3, Section 17. Follow the procedures in DoDM 5200.01-Volume 4, Information Security Program: Controlled Unclassified Information for controlled unclassified information.

TABLE OF CONTENTS

2 DOCUMENT CONTROL

REVISION HISTORY				
REV LTR	**Rev**	**Date**	**Author**	**Document Changes**
A	1.0	08/30/2023	Stephen Mitchell	Initial Draft
A	1.01	09/01/2023	Stephen Mitchell	Layout, Format, Editing. Content Review and Correction. Structural Analysis and Enhancement.
A	2.00	09/22/2023	Stephen Mitchell	Final Draft

Table 1: Revision History

2.1 APPROVALS

NAME	TITLE	ROLE	SIGNATURE	DATE
John Rizzo	Sr. Vice President, Business/Programs	Program Manager		09/22/2023
Kleber Valencia	Vice President, Engineering	Design Architect		09/22/2023
Everett Sellner	Software Development, Engineering	Software Architect		09/22/2023
Scott Goeringer	Quality Manager	Quality Control		09/22/2023

Table 2: Approvals

2.1 POINTS OF CONTACT

NAME	TITLE	EMAIL	CONTACT NUMBER
John Rizzo	Sr. Vice President, Business/Programs	jrizzo@sagrad.com	
Kleber Valencia	Vice President, Engineering	kvalencia@sagrad.com	
Tony Crosthwaite	Vice President, Operations	tcrosthwaite@sagrad.com	
Scott Goeringer	Quality Manager	sgoeringer@sagrad.com	
Martha Harriman	President	mharriman@sagrad.com	

Table 3: Points of Contact

2.1 SERVICE DESK

In the event of a question about the operation of the systems, users should contact the Sagrad Service Desk at +1 (321) 726-9400, or via email: sales@sagrad.com

3 ACRONYMS AND ABBREVIATIONS

ACRONYM	DEFINITION
AFTU	Automated Flight Termination Unit
ATE	Automated Test Equipment
BOM	Bill of Materials
BIT	Built-In-Test
CMMI	Capability Maturity Model Integration
CASS	Core Autonomous Safety Software CASS
CDR	Critical Design Review CDR
DTD	Defect Tracking Database DTD
ECN	Engineering Change Notice
HDS	Hardware Design Specification
IV&V	Independent Verification and Validation
MDF	Master Development Folder
MDN	Master Documentation Node
MVC	Master Validation Checklist
OCM	On-Chip Memory
PDR	Preliminary Design Review
PRR	Production Readiness Review
RSE	Responsible Software Engineer

ACRONYM	DEFINITION
SDS	Software Design Specification
SIR	Software Implementation Review
SRS	Software Requirements Specification
SRB	Software Review Board
SCM	Source Control Management
SDR	Specification Development and Review
SOW	Statement of Work
SFR	System Feasibility Review
SOC	System on a Chip
TRR	Technical Readiness Review

Table 4: Acronyms and Abbreviations

4 INTRODUCTION

This document proposes a software development process and software assurance plan for the **AFTU** and **IAFTU** software development. Documenting our practices and procedures in software engineering will also enable us to improve our position on the scale described by the Capability Maturity Model Integration (CMMI).

The steps of the software development plan (SDP) outlined below are a refinement of QOP 8.3 Product Design and Development Plan. In this regard, the SDP fills in the details of the high-level QOP 8.3 as they apply specifically to software engineering. Though additional steps and details are added to this document, it must conform to the master plan defined by QOP 8.3.

The remainder of this document specifies a simple development process that is geared toward providing the foundation for more advanced forms of process control necessary for safety critical systems while at the same time remaining mindful of the practical need that it is achievable by a small organization with stringent demands concerning efficiency and time to market.

This document is split into three sections in addition to the introduction.

- Objectives of the Development Process

Why are we defining this process, and what must it achieve?

- Process Management

All information related to the process, such as specifying the required documentation.

- The Software Development Process

A description of the complete process as applied to software development.

5 OBJECTIVES

Many organizations attempt to create a quality process to prevent defects and control the stability of their product. Not all these efforts are successful. Success and failure in establishing a development process revolves around whether it is followed. More importantly, it is fair to ask whether an organization has any way of knowing when a process is being followed.

The following questions have specific answers in organizations that successfully implement any systematic methodology to control product development.

1. How does the company know which process steps have been followed for a release?
2. Is the process taken for granted, or are checklists used to prove each step?
3. Are metrics of any kind collected and used to improve performance?
4. Is the documented process state of a release checked into SCM along with the release?
5. How are employees trained in the process?
6. Are employees rewarded for attentiveness to defect prevention?
7. Who in the organization is responsible for enforcing the process?

All these questions spring from an even more fundamental one.

How sophisticated is the organization's attitude toward human error in general? Is there a basic understanding in the organization that software tends to be defective until proven functional, or is it optimistically assumed to be functional unless proven defective?

5.1 DEVELOPMENT PROCESS OBJECTIVES

This document has been created to facilitate the following specific objectives.

1. Provide a repeatable and provable software development process.
2. Ensure that customer needs are met in all respects when software is delivered.
3. Provide the link between QOP 8.3 and specific software development practices.
4. Identify and prevent software defects before release.
5. Ensure that our software development process is suitable for external review.

6. We need to move a greater degree of our capability from individuals to company intellectual capital.
7. Provide a mechanism for identifying specifically what steps are completed per release.
8. Start collecting metrics and implement a mechanism for continuous improvement.
9. Specify the tailoring of RCC319 as a defined and repeatable process.

5.2 OUTPUTS

This document must specify the following outputs.

1. A document specifying the entire software development process itself (this document).
2. A list of checklists used to control software delivery based on conformance to the process.
3. Instructions for correlating process directives and results to each software release.

5.3 SYSTEMATIC SOLUTIONS

One of the primary goals of this process is to enable the organization to fix problems systematically rather than episodically. Fixing problems individually as they occur without regard to their underlying case is not indicative of a mature development process. An effective development process identifies individual errors as symptoms of a deeper procedural problem that permitted the problem to occur in the first place.

From this perspective, a problem is not solved to Sagrad's satisfaction until a test or inspection is added to the development process, which would have exposed the problem in the past before it occurred and will be exposed again in the future. Furthermore, the process should be adjusted for each defect to locate the problem internally during development prior to external validation. This feedback mechanism radically increases the probability that Sagrad will catch a destructive defect during development and testing.

5.4 EVIDENCE AND PROVABILITY

This document only concerns product development activities that an external observer can validate. Many of the industry standard techniques in software development depend heavily on the integrity and

professionalism of the engineers building the product. Such activities are important and should continue; however, such practices are not specified here unless their use can be validated.

For this reason, we require a paper trail for all steps in this process and the ability to archive such artifacts with each software release. Otherwise, there is no way for an external observer to know if the product itself has been developed using this process. This document goes a step further than QOP 8.3 by moving from the statement of procedural steps to defining the documentation that provides the evidence for each step.

5.5 THE DEFINITION OF REQUIREMENTS

This document defines the word '**requirement**' as a statement of intent that inspection can functionally test or validate. Concerning general statements of intent, for instance, in creating a statement of work (SOW), we will not use the term 'requirement' in this document.

To keep terms clear, we use the following terms to distinguish between statements of intent at various junctures of the process.

- **Capabilities and Objectives**

  Capabilities and objectives are critical high-level goals that define the System's behavior before the phase in which specific functional requirements can be created. Typically, functional objectives are defined in the **Statement of Work** (SOW). Some, but not all, of these objectives are destined to become testable functional requirements later in the development process.

- **Functional Requirements**

  Requirements are testable or inspectable statements of intent that define the functional behavior of the product in sufficient detail to implement it. Since these requirements are formulated during the requirements phase after the Statement of Work (SOW), the Statement of Work, by definition, cannot be said to contain these functional requirements.

- **Regulatory Requirements**

  Regulatory requirements are defined in external standards, such as RCC-319, which define safety precautions criteria for stability and determine how the System must operate and how it is developed, but which may or may not contain requirements about specific functional behavior.

  This distinction is necessary since many of the requirements in RCC319 play a critical role in determining how dependably the System operates but not necessarily what it does. Regulatory requirements are referred to here as a type of requirement since they can be tested or inspected for validation.

5.6 WATERFALL DEVELOPMENT

Another objective of the plan is to incorporate the classical **waterfall** development model into an engineering environment highly contingent on regulatory requirements, where virtually every development step is heavily reviewed internally and externally.

It is helpful, therefore, at this point to review the **waterfall development** cycle as it applies specifically to our development needs. The model generally consists of several primary phases performed on the entire software system. Briefly, these steps are as follows.

1. Envision the product based on demand.
2. Specify functional requirements.
3. Design the product.
4. Implement the design.
5. Test and validate requirements.
6. Deliver and maintain the product.

Each step in this process is concluded with a review before proceeding to the next step. The following steps apply the **waterfall** model to a typical aerospace scenario.

1. Envision the product-based industry demand and regulatory standards.
2. Work with customers to define functional objectives in the statement of work.
3. Estimate the development effort and agree on commitments.
4. Analyze all operative regulatory requirements.
5. Specify all functional requirements.
6. Create and document the design.
7. Specifying the structural architecture of that design.
8. If necessary, revise commitments based on the details provided by requirements.
9. Implement the source code.
10. Perform code reviews, static analysis, and other non-functional inspections.

11. Perform functional unit tests.
12. Perform functional system tests.
13. Perform internal requirements verification and validation.
14. Submit the software and all artifacts for external independent verification and validation.
15. Prepare the product for production and deploy it.
16. Maintain the product through defect fixes and additional features.
17. Analyze the entire process to glean improvements for the subsequent development effort.

We assume that each of the phases listed above is punctuated by a peer review of the artifacts generated at that stage, so these steps are not listed here separately.

Various forms of **Agile** development are not used in their purest form here because a flight termination system, for the most part, requires a fully functional set of subsystems to operate. The majority of these must be in place before the first functional test can be executed. In this context, incremental delivery of the entire System at the short intervals defined by **Agile** is normally impractical.

Nevertheless, incremental delivery plays a vital role in our process, especially near the end of development, because often unvalidated system prototypes can be provided to clients for testing before the validated version. Keep in mind unvalidated does not mean uninspected since Sagrad has a formal and extensive internal review process active between the listed steps, executed before prototypes are provided.

One significant adjustment Sagrad makes to the **Waterfall** method is that we employ all test activities in parallel with development so that final test runs are formal yet uncomplicated.

6 PROCESS MANAGEMENT

This section covers the management of the process itself and specifies the required documentation.

6.1 PROCESS DOMAINS

An effective development process cannot be reduced to a single linear series of events but instead requires that multiple tracks of activity over several domains are all carefully orchestrated to operate in parallel. These domains are as follows.

- **Product Development**

  Product development is executed by engineering and covers all the components of the process related directly to building the product, which includes specifying requirements and designing the System that implements them.

- **Schedule Estimation**

  The ability to perform accurate schedule estimation falls under the domain of process control since the date a product is delivered is usually as important as what is delivered. For this reason, a significant component of the research done in the development process area focuses on increasing the accuracy of schedule estimation.

- **Configuration Management**

  Configuration management covers all facets of system integration to ensure that a system's correct components are documented, assembled, stored, and identified for each software release. It also ensures that the state of the System is defined and can be maintained.

- **Defect Tracking**

  A critical process component ensures that defects and issues are recorded and managed correctly. In addition, the link between defect reports, the code that fixes them, and the tests that validate the fix must all be known, traced, and documented.

- **Metrics and Defect Prevention**

  This area covers the implications of defect tracking by recording all data related to issues so that the implications of this data can be analyzed. Examples are identifying what code sections are problematic and how entire classes of defects can be prevented.

  - **Requirements Validation and Traceability**

  In a safety-critical system, every requirement must exist in a logical chain that connects the original requirement and all the steps necessary to validate it. This includes demonstrating the connection between the requirement and any potential tailoring performed on it, its implementation, and ultimately to the unit and/or system test that validates it, as well as the proof delivered to **IV**&**V** that it has been validated.

  - **Process Management and Continuous Improvement**

  The development process itself is a product that must be designed, debugged, maintained, and improved.

  - **Independent Verification and Validation**

  Everything in all other domains must contribute to the central goal of ensuring the System can be proven externally. This document seeks to make **IV**&**V** the central goal of the entire development effort rather than just another test activity.

The process related to each of these domains is specified in the sections below. Ultimately, this document will define processes for each of these domains. Currently, for the first revision of this process, we define only the product development cycle and requirements traceability.

6.2 THE MASTER VALIDATION CHECKLIST

The **Master Validation Checklist** (MVC) is a spreadsheet listing and tracking every process step. A unique copy of the MVC is created and assigned to each product development program.

The checklist is updated once the steps for a particular software release are executed. Before the software is released, the MVC can be checked to see what process steps have been executed for the release and whether any critical steps are missing.

It is important to note that cursory updates to this list should not be permitted, and items should not be marked as accomplished unless a person has specific knowledge that the task has been completed. It is far better to release a product with some items accurately left unchecked so that post-mortem analysis is accurate than it is to check all boxes in a perfunctory manner to expedite a release.

6.3 MASTER DEVELOPMENT FOLDER

To remove all ambiguity regarding the association of documentation to a particular product release, we use the concept of a single point of entry and require a single development folder to be assigned to a product, referred to as the **Master Development Folder** (MDF). All official documentation associated with product development must be stored somewhere under this folder or connected to it specifically by a reference in the document spreadsheet.

This concept of 'single point of entry' is ubiquitous in engineering at every conceivable level and is used, for example, in user interfaces, web page design, the architecture of task message queues, and in security design such as airport security lines. No process will remain active if there is any complexity whatsoever in the ability of personnel to quickly locate all documentation of any kind associated with a project. The master project folder is a straightforward way to resolve this.

A complete copy of this folder must be archived with the software release in source control so there is no question regarding the state of the process at the time of release. For this reason, care must be taken to ensure that the **MDF** does not contain excessively large files or folders. Therefore, tools folders, installation folders, or any build outputs should not be placed in the **MDF**.

The latest version of any document related to the software must be placed in the tree under this root folder, and an entry must be placed in the master documentation tree in the root of the master documentation node.

Files such as this development process are placed in the **MDF** close to the root folder, and files that apply only to a particular product are placed in subfolders. Critical files such as QOP 8.3, which must have the original copy in the quality folder, are listed in the **MDF** document tree with a reference and document version number.

At the very least, the **Master Documentation Node** (MDN) should contain the following documents.

1. The **Software Documentation Tree** as a spreadsheet in the **MDF root** folder.

2. The official copy of the **Master Validation Checklist** for each product release.
3. The official copy of any regulatory standard, for instance, RCC319-19T.
4. The requirements traceability spreadsheet.
5. The current copy of each project's design specifications, test plans, and test results.

Suppose the official copy of a document must exist at another location, for instance, in the network Quality folder. In that case, the Master Documentation Node must contain a link to that location in place of the document itself.

6.4 TRACEABILITY

Several of the process domains listed here require traceability. Traceability refers to the specification of logical links between items in a process. The most important one in this context is the link between requirements and validation tests. This traceability requires a chain through several areas with top-down and bottom-up links in both directions conceptually.

These relationships must be reproducible in a tangible form when an external review is requested by any outside party, such as **IV&V** or another organization's quality and configuration management department.

The first step in implementing traceability in this new process is to recognize some common pitfalls to implementing solid requirements validation traceability. The most common mistake occurs when a probabilistic approach is employed during validation traceability. When engineers are asked to fill in such a traceability matrix for many requirements, filling in system test cases that 'probably' exercise that requirement is tempting. The following logic is used.

- *'This test could not pass unless this requirement was implemented; therefore, this test validates the requirement.'*

This deductive method is imprecise and not indicative of a mature development process. Instead, each component test must be specifically written to validate a specific requirement. It is permissible for a single test to validate multiple requirements if it was explicitly designed to do so. The practice of hunting for existing tests to validate a lengthy list of requirements with little effort should not be permitted in a serious quality process.

A viable quality process requires a valid link between each requirement and a specific individual test designed to validate that requirement, which exists within the larger framework of a test suite such as the Sagrad run for the record.

In addition, the test itself should clearly print the test results along with the requirement number so that the final inspection of the test run can know with certainty which requirements can be checked off as validated.

Listing all tests, assuming they validate the requirements, leaves outside parties, such as **IV**&**V**, with no information regarding the validation state of a requirement. Instead, the following System will be employed.

6.5 REQUIREMENTS VALIDATION TRACKING

Sagrad employs the following documents to accomplish requirements validation traceability in both directions. The specific network links to these documents will be provided in the next version of this document. Currently, all these documents are used by our **IV&V** partners to validate our product revisions.

TRACEABILITY POINT	DOCUMENT	NETWORK LOCATION
Regulatory Requirements	RCC-319-19	
Requirements Link to Tests	System S.R.S. Spreadsheet	
Specification of Tests	Run for Record Test Suite	
Result of each Test	Test Results HTML	
Final Proof of Validation	Run for Record Publication	

Table 5: Requirements Validation Tracking

6.6 TRACEABILITY PHASES

This section describes a standardized method for increasing the level of detail in requirements traceability so that an organization such as Sagrad can identify its current phase of traceability and pursue a predictable path toward fine granularity in traceability. The ultimate goal for this level of granularity is phase three, which stipulates one-to-one automated traceability from individual requirements to individual tests and includes fully automated test output for all validated results. This end goal increases the efficiency of inspection by partners, ranges, and **IV&V**.

Traceability can be performed at essentially four levels of detail, and normally, organizations start at phase zero and progress to phase three, which equates approximately to a **CMMI** level of one to level four (Capability Maturity Model Integration).

TRACEABILITY PHASE	TYPE OF VALIDATION TRACEABILITY	APPROXIMATE CMMI LEVEL
Phase 0	No Validation Traceability	Level 1 'Initial'
Phase 1	Test Plan Traceability	Level 2 'Managed'
Phase 2	Individual Test Identification	Level 3 'Defined'
Phase 3	Fully Automated One-To-One	Level 4 'Quantitative'

Table 6: Traceability Phases

In **phase 0**, there is no procedural traceability, and requirements are validated once per release without following a defined procedure.

In **phase 1**, traceability is proven to be satisfied by an entire test plan defined by the process and verified to have a high probability of validating all requirements. Individual requirements, however, cannot be traced to any particular part of the test in the documentation or the printed test results without examining the test on a case-by-case basis.

In **phase 2,** traceability is repeatable and part of a validated procedure in which individual requirements are proven to be validated by specific tests. Full traceability in both directions, top to bottom and bottom to top, can be validated externally by regulatory bodies by connecting individual requirements to one or more separate tests explicitly designed to validate them.

In **phase 3,** traceability is both repeatable and fully automated and documented. At this phase, every requirement can be connected to every test using documented procedural traceability in both directions. As in phase 2, traceability is part of the process and is easily inspected by regulatory partners. In addition, the System is fully automated, so each **System** test prints out specific requirement numbers and a pass/fail result for every test so that independent validation partners can inspect the entire test suite.

Sagrad currently employs both **phase 2** and **phase 3** traceability. Sagrad software system test employs a Run for Record, which employs phase 2 traceability between all regulatory requirements and test results. This is documented and validated externally by **IV&V**. Phase Sagrad uses **phase 3** traceability for **Software Operating System Validation** tests. Ultimately, Sagrad plans to employ **phase 3** traceability for all externally visible tests.

7 THE SOFTWARE DEVELOPMENT PROCESS

This section prescribes the steps in our product development process executed by software engineering. The master process defined in QOP 8.3 has been adjusted here for software development needs. However, software development must be managed so that the corresponding elements of each process align at important junctures to ensure that software is available to support hardware during the entire system development effort.

7.1 PROCESS TABLE

We start by listing the steps alone in a table, and the far-right column shows the equivalent step in the master process defined by QOP 8.3.G. Process steps that branch into a parallel activity during the software implementation phase are marked 'parallel.'

STEP	ABREV	PROCESS STEP DESCRIPTION	PARALLEL	QOP 8.3.G
1	SOW	Statement of Work	Linear	7.1
2	SPP	System Program Plan	Linear	7.2
3	SFR	System Feasibility Review	Linear	7.3
4	TRR	Tailoring of Regulatory Requirements	Linear	
5	SRS	Software Requirements Specification	Linear	7.5
6	SDS	Software Design Specification	Linear	7.6
7	PDR	Preliminary Design Review	Linear	7.8
8	CDR	Critical Design Review	Linear	7.10
9	SNG	System Go/No Go	Linear	7.14
10	SIP	Software Implementation Phase	Linear	7.11
11	SCR	Software Code Review	Parallel	

STEP	ABREV	PROCESS STEP DESCRIPTION	PARALLEL	QOP 8.3.G
12	SSA	Software Static Analysis	Parallel	
13	SUT	Software Unit Test	Parallel	
14	SST	Software System Test	Parallel	
15	SIR	Software Implementation Review	Linear	
16	RFR	Run for Record	Parallel	
17	PRR	Production Readiness Review	Linear	7.15

Table 7: Process Table

7.2 SOFTWARE PROCESS STEPS

This section explains each step in the software development process. Unless otherwise noted, the steps should be executed in the order in which they are listed.

7.2.1 STATEMENT OF WORK (SOW.)

Defines the product's high-level functional and performance objectives and identifies the source of all regulatory standards. See: (QOP 8.3.G-7.1).

7.2.2 PROGRAM PLAN

The program plan should establish and document all commitments to shareholders regarding the externally visible development schedule and its deliverables. This includes the delivery of prototypes, the production schedule, and shareholder reviews. A review of the program plan by the appropriate shareholder is part of this step and must be completed. See: (QOP 8.3.G-7.2).

At this point, the customers' requirements for delivery timing must be carefully analyzed to see if the product development schedule can accommodate them.

7.2.3 SYSTEM FEASIBILITY REVIEW (SFR)

The software **System Feasibility Review** (SFR) is the equivalent of the hardware **Specification Development and Review** (SDR) outlined in QOP 8.3.G-7.3. This review occurs shortly after the SOW and Program Plan are completed. It is provided to the customer as the first opportunity to examine and validate the viability of the program as well as the high-level capabilities of the product.

This review is essentially a reality check focusing on the high-level block diagram of the System. The purpose is to validate new core technologies, essential capabilities, and significant new features. The challenge of the review at this juncture is that requirements and design specifications have not yet been written and are, therefore, unavailable. However, much is known about the project and should be captured and presented in a first draft of the System or hardware design specification.

For instance, at this point in the program, it is normal for major new system components to have been selected, such as the mechanical size, processor technology, the **System on a Chip** (SOC), new component flight software versions, major new features, and the high-level system organization. All this can be presented to the customer. The most expedient technique is to place this information in the first chapter of the system hardware design specification since it will end up there anyway.

7.2.4 TAILORING OF REGULATORY REQUIREMENTS

The existence of this step depends on the specific program. If requirements for the product are determined to any extent by regulatory requirements, then this step is necessary. Regulatory and compliance requirements must be tailored before customer functional requirements can be defined. This must precede the specification of functional requirements since the latter is highly dependent on the former. An example of tailored input is RCC319-19T.

The output of this process should be a single spreadsheet with columns showing the original state of the requirement and the version of the requirement that will be operative in the product during development. All requirements must have a unique identifying number that never changes and is not dependent on a spreadsheet row. Typically, the original number identifying the requirement for a tailored spreadsheet is sufficient.

7.2.5 SOFTWARE REQUIREMENTS SPECIFICATION (SRS.)

The software requirements specification has now been written and delivered for peer review. All three major disciplines must generate their own requirements specifications at this juncture. This will include mechanical, electrical, and software, depending on the project. Since the three are contingent and the software requirements highly depend on the hardware design, the hardware requirements specification is typically completed first. It is permissible to combine the system requirements specification and the hardware requirements specifications into a single document.

There are three inputs into the software **SRS**. First is the statement of work (SOW), which contains the capabilities and performance objectives; second is the regulatory requirements tailoring and additional functional requirements requested explicitly by the client, which are not yet captured in the previous two documents. The **SRS** must be a separate document and can refer to a version of the Regulatory Tailoring to identify those requirements rather than repeating them verbatim.

7.2.6 SOFTWARE DESIGN SPECIFICATION (SDS.)

The software design specification is created. The **SDS** describes the structural architecture of the System and includes whatever block diagrams are necessary to communicate the design. At the very least, the **SDS** should describe the software infrastructure, including all the system tasks (threads), how the operating System is used, and how the major classes and objects are organized.

The SDS should specify the design of the software in sufficient detail so that different development teams possessing only the SDD can implement the System with adequate accuracy and similarity so that the specified requirements are met and validated in both cases. Detail beyond this level is not required.

At this juncture, all disciplines must generate their design specification. Depending on the project, this will include mechanical, hardware, and software design specifications. It is permissible for the system and hardware design specifications to be covered in the same document. The **HDS** and **SDS** must pass peer review before proceeding.

7.2.7 PRELIMINARY DESIGN REVIEW (PDR.)

A **Preliminary Design Review** (PDR) is now conducted on the hardware. Customer approval is required to proceed. An internal and external review as part of the **PDR** can be conducted if required. The program manager can also skip the internal review and move directly to the external **PDR** if the state of the hardware and system design specifications do not warrant an internal review.

It is not usually necessary to perform a separate **PDR** for the software. This is because the **PDR** and **Critical Design Review** (CDR) reviews are back-to-back in this process, and only a single review is required for the software. The software design, however, can be included in the hardware **PDR** if that information is needed to review the system design as a whole during the review.

7.2.8 CRITICAL DESIGN REVIEW (CDR.)

A **Critical Design Review** (CDR) of the software design is conducted to review software design specifications in their current state before the implementation phase. Customer approval is required to proceed.

A review of all components of the product design is now possible since these design specifications are complete. A full review of them by the customer is appropriate to ensure that the product will meet their expectations before the implementation of that design starts. Action items are entered into the defect tracking system. (QOP 8.3.G-7.1).

The software **CDR** should occur in the same time frame as the hardware **CDR**, and the two can be conducted in a single **CDR** if the program manager wishes to review the entire system design in the **CDR**. The hardware **CDR** is held before purchasing printed circuit boards and other components to ensure the design is correct before committing to acquire elements of the **bill of materials** (BOM.)

The **CDR** occurs in two phases. Typically, two reviews are conducted, the first being an internal review of all the relevant requirements and design specifications and the second a presentation of the same material to external customers and shareholders for their approval. The program manager can bypass the internal **CDR** if the design is sufficient to proceed directly to an external review. Following the **CDR**, the quality and configuration management department should account for all requirements and design specifications and assign a configuration management SG number.

7.2.9 SYSTEM GO/NO GO

External shareholders now have an opportunity to make a **go/no go decision on the project after examining** the final design in detail. Once they approve the design, the implementation phase can begin. See: (QOP 8.3.G-7.14).

7.2.10 SOFTWARE IMPLEMENTATION PHASE

Now that all requirements and designs have been reviewed and approved, the software team can begin implementing the source code. During the implementation phase, the software source code is written. See: (QOP 8.3.G-7.11).

Software should be compiled as written to ensure it can always be built. The written code should be checked and pushed into source control at least once a day as it is written. Preferably, the code stored

in source control should always be in a fully buildable state. When the code cannot be built, it is a higher priority to check it into source control than waiting until it can be fully compiled.

7.2.11 SOFTWARE CODE REVIEW

Peer reviews are conducted on the source code as it is written. All source code is typically reviewed twice. All new classes, or all new source code files, are peer-reviewed shortly after they are written. Then, all the source code is reviewed, including all the individual classes and files, when implementation is complete during the **Software Implementation Review** (SIR).

After reviewing a particular source code file, a check mark, including the date and **GIT Source Control Version** tag, should be entered into the code review checklist in the **Master Documentation** node.

7.2.12 SOFTWARE STATIC ANALYSIS

Once the software implementation phase begins, the static analysis effort begins in parallel shortly after the start of implementation. As time and manpower permits, static analysis should be run constantly on the current software revision as the software is implemented.

The official static analysis results document should be updated to reflect the latest results as each static analysis run is performed. Static analysis violations should be resolved by entering them into Redmine's defect tracking system and informing the software engineer that the code needs to be changed to address the issue.

Since the number of static analysis violations can be exceedingly high, entire classes or collections of violations can be accounted for in a single defect tracking issue. To expedite the process, a separate branch of the code in source control can be created in which the static analysis engineer is free to make fixes to each violation, and at a later date, this branch of the code can be merged into the master development branch by the senior software engineer responsible for the source code.

7.2.13 SOFTWARE UNIT TEST

Once the software implementation phase begins, the unit test effort begins in parallel on or shortly after the start of code implementation. 100% code coverage is required on the source code.

The output of the unit test process is the unit test folder, which contains all the unit test files and the **HTML** output of each test. This folder should be maintained and updated after each unit test run and periodically checked into source control along with the source code.

7.2.14 SOFTWARE SYSTEM TEST

The software system test is implemented and conducted in the form of a **run for record**. This is a suite of automated tests that use multiple flight patterns to test every facet of the software in nominal and termination cases. As time and manpower permits, the **run for record** should be run at regular intervals in parallel with software implementation to address issues as the code is being written.

7.2.15 SOFTWARE IMPLEMENTATION REVIEW (SIR.)

After software implementation is complete, all software artifacts must be accounted for and checked into source control in preparation for the **Software Implementation Review** (SIR). This includes the following items: the software source code, the static analysis results document, the unit test folder, and the HTML output files.

A final code review is performed on all the source code during the Software Implementation Review. In addition, critical modules such as the main task loop and **CASS** interface are reviewed in a group setting during a code review meeting. All the artifacts listed are also reviewed for completeness and accuracy. The results of the SIR are recorded in the software **Master Validation Checklist** (MVC) and checked into source control along with the software source code.

7.2.16 RUN FOR RECORD (SIR.)

A final **run for record** is performed on the software to establish the official system test results for the release. After completing this, the results are entered into the software **Master Validation Checklist** (MVC).

7.2.17 PRODUCTION READINESS REVIEW

The **Production Readiness Review** (PRR) for the software is performed as part of the **PRR** executed for the entire System as part of the hardware development effort. The software **PRR** must support the overall System **PRR**, as described below.

A production readiness review is conducted, and if the unit is ready for production, engineering releases the project to manufacturing. From this point, an **ECN** (SG932-0127 Engineering Change Notice) is required for any document changes. QOP 7.5 Documented Information describes the ECN process. See: (QOP 8.3.G-7.15).

During the software PRR, the **Software Master Validation Checklist** (MVC) is reviewed during a meeting dedicated to this purpose to see if any missing elements prevent the software's release. The software release is pronounced ready for production if the **MVC** is complete. The **MVC** is updated with this fact and checked into source control along with the source code.

7.3 SOFTWARE DEVELOPMENT PROCESS DIAGRAM

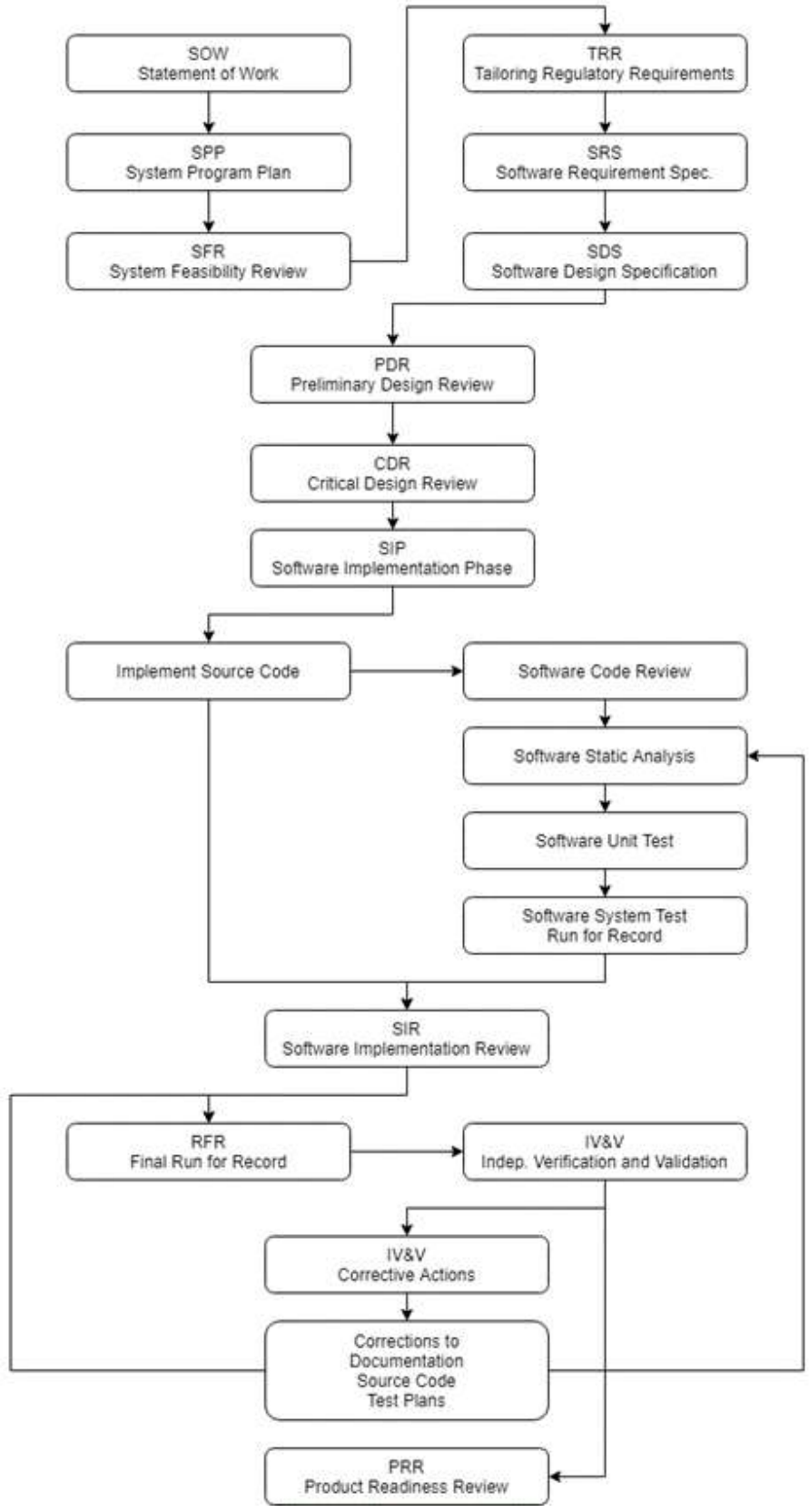

Figure 1: Software Development Process

7.4 SOFTWARE MAINTENANCE PROCESS

This section describes Sagrad's process for software maintenance and defect tracking.

Sagrad uses this procedure to identify and track all product development and maintenance issues. This process is applicable for both trouble reports and new features. In addition, it should also be used for technical questions if such a question requires a significant amount of development work in testing or analysis, even if a code change is not required.

The engineer should refer to QOP 8.3.6 Software Change Request Process if an issue requires a software change. The following procedure covers all aspects of a software change request and defect resolution, including ones not mentioned in QOP 8.3.6.

7.4.1 RECORD ALL ISSUES IN REDMINE

This first step is a critical point of general strategy that is always operative. The primary tool used for all software issues is our **Defect Tracking Database** (DTD), specifically the tool Redmine. This network-accessible application provides all the fields necessary to capture, track, and resolve issues at any time in the development process. We use Redmine for all issues, including software bugs, hardware problems, configuration and production issues, quality process control issues, and questions.

Suppose problems or questions arise at any point in the product development process, and there is any doubt about which document or source code file is appropriate to capture the problem statement. In that case, entering the issue into Redmine is always a safe strategy. Once this is done, the information can be acted on appropriately later.

7.4.2 IDENTIFY THE PROBLEM OR FEATURE

Problems can be reported from a host of sources, both internal and external.

- Customer sights. Insights?
- IV&V Product Review Partners.
- Our own manufacturing and production floor.
- From within engineering during testing.
- From within engineering while editing source code for other reasons.

- Development partners such as DOD and CASS.
- From the test group during unit testing, static analysis, or system test.

Changes may be necessary when engineers determine that the software contains an error or is missing a feature after a major release. Errors may include faults in design or implementation, incomplete requirements, incomplete validation or testing, and errors in documenting the requirements or design. See: (QOP 8.3.6.B-7.1).

7.4.3 COLLECT ALL INFORMATION

It is critical to collect actionable information when any defect is reported. The more specific this information, the faster the problem can be resolved. At the very least, a complete description of how to recreate the problem must be documented.

- Who reported the issue?
- The date the problem was first observed.
- The severity of the issue from the client's perspective.
- When does the client need a fix delivered?
- What hardware and software release versions did the problem occur in?
- Detailed instructions on how to recreate the problem.
- To whom should the issue be assigned?
- What are the long-range implications of the problem on our product base or technology, if possible?

7.4.4 CREATE A REDMINE ISSUE

The employee should log into Redmine and create a new issue. New issues should be entered into the **SCR Database** section. The above information should be included, and the following fields should be filled for a new defect. Tracker (Issue Type), **Subject**, **Description**, **Status** (New), **Priority**, **Assignee**, **Category**. Redmine assigns each issue a unique number for identification and tracking. (QOP 8.3.6.B-7.2). Required fields are also listed in (QOP 8.3.6.B-7.4).

7.4.5 SOFTWARE REVIEW BOARD MEETING

The **Software Review Board** (SRB) meets regularly to identify and discuss outstanding Redmine issues. A decision is made during this meeting that action needs to be taken on an **SCR** and that it should be assigned to an engineer for analysis and resolution. See: (QOP 8.3.6.B-7.4).

7.4.6 FETCH THE SOURCE CODE FROM GIT VERSION CONTROL

Assuming the issue is handled by software, the developer should fetch the latest version of the current development branch from source control, in this case, **GIT**. Typically, the appropriate branch will be the master branch. If development is temporarily working on a branch other than the master, then that branch's latest version should be acquired.

7.4.7 CREATE A GIT VERSION CONTROL BRANCH

Suppose it is anticipated that the solution to the defect will take significant development or experimentation. In that case, a new branch should be created from the existing master branch or development branch to insulate this work from the rest of the code.

- git pull
- git branch -r
- git checkout branch_name

7.4.8 ANALYZE THE PROBLEM AND CONSTRUCT A SOLUTION

The problem should be analyzed, and the appropriate solution constructed. If the solution to the problem requires one or more source code alterations, they should be implemented at this point. The new software code should include comments where appropriate so that both the problem and methodology behind the solution are clear to anyone maintaining the code.

7.4.9 COMMIT INITIAL SOLUTION TO GIT

After the solution is implemented in the source code, it should be checked into source control. It is always best to do this even before the solution is tested. This is because it is not uncommon for some problems to require some degree of experimentation, multiple builds, and test runs. Keeping an account in source control of the attempted solutions often pays off when they need to be recalled. Ensure the comment includes the **Redmine SCR** number and briefly explains the fix. For example:

```
commit -m "Fix for SCR 517 main loop index limit off by one resolved."
```

7.4.10 UNIT TEST THE SOLUTION

The proposed solution must now be unit-tested to confirm that it resolves the issue.

7.4.11 COMMIT FINAL SOLUTION TO GIT

After the fix has been validated, the final solution implemented in the source code should be committed to **GIT**. If a source control branch was created for the issue, it should now be merged back into the branch from which it was divided. This includes merging the solution into the master branch if necessary. The solution in its current and potentially final state should be committed to **GIT** so that the **GIT** tag, which identifies the software being reviewed, can be referred to specifically for the code review.

7.4.12 CODE REVIEW THE SOLUTION

A peer review must be scheduled and executed to review the code change before release. Refer to the appropriate quality procedure **QOP** for the precise steps in executing a peer review. The code review checklist should be used for a defect solution code review. Please refer to 'QOP 8.3.4 Peer Review Process' to execute this step. Code review findings should be entered into Redmine using the notes section of the original **SCR**. The **GIT** tag of the software version being reviewed must be entered along with all information pertaining to the review.

7.4.13 IDENTIFY PROCEDURAL FIXES FOR CONTINUOUS IMPROVEMENT

As part of the code review, the team should carefully consider if the problem occurred due to an oversight in the development process or if the process can be updated in such a way that it would have identified the issue before the release of the software. Commonly, this occurs as a simple addition to the code review checklist.

If this is the case, enter an **SCR** for the process improvement, and in it, refer to the **SCR** number for the original defect and explain the procedural improvement that would have resolved the problem. Also, inform the quality department that a **QOP** quality procedure is to be updated.

7.4.14 SOFTWARE LEAD VERIFIES RESULTS

The software lead should verify everything has been implemented correctly and that all code has been committed and pushed to the appropriate source control branch. All necessary unit and system tests have been performed. See: (QOP 8.3.6.B-7.10).

7.4.15 COMMIT ADDITIONAL CODE TO GIT

If the source code needs to be edited again due to the code review or software lead inspection, then an additional commit to GIT may be necessary.

7.4.16 UPDATE THE REDMINE SCR

Upon finalizing the source code in **GIT** and completing the code and software lead reviews, the original Redmine SCR should be updated to capture the resolved state of the issue. This final update should incorporate insights and information from all three stages to provide a comprehensive overview of the solution. Refer to QOP 8.3.6.B-7.9. After the work is completed and any related source code is checked into **GIT**, the **SCR** is transitioned to the resolved state to await verification by the lead responsible for the **SCR**."

7.4.17 SUBMIT SCR VERIFICATION FORM

After these steps are performed, the **Responsible Software Engineer** (RSE) will fill out an **SCR** Verification Form (SG932-0249). The **SCR** lead must review and sign this form to verify that all steps have been completed. (QOP 8.3.6.B-7.11)

7.4.18 PROCEED TO THE RELEASE WORK INSTRUCTION

At this point, the resolution is ready to be released and deployed. The work instructions for releasing software depend on the software product being released; this document does not cover that. For instance, **AFTU** and **IAFTU** software may require a product readiness review, a **run for record** system test, software delivery, and all regulatory and functional requirements specifications to the **IV&V** team for independent verification and validation followed by deployment.

For more information on the release process, please see the following documents.

- QOP 8.3 Product Design and Development Rev G 12-13-2022
- QOP 8.5.5 Recertification process Rev - 4-15-21.pdf

- QOP 8.3.6 Software Change Request Rev B 4-21-23
- SG932-0206 AFTU Configuration Data Report Rev F 7-20-2022.docx

7.5 SOFTWARE MAINTENANCE PROCESS DIAGRAM

The following workflow diagram shows a graphical view of the above software maintenance process.

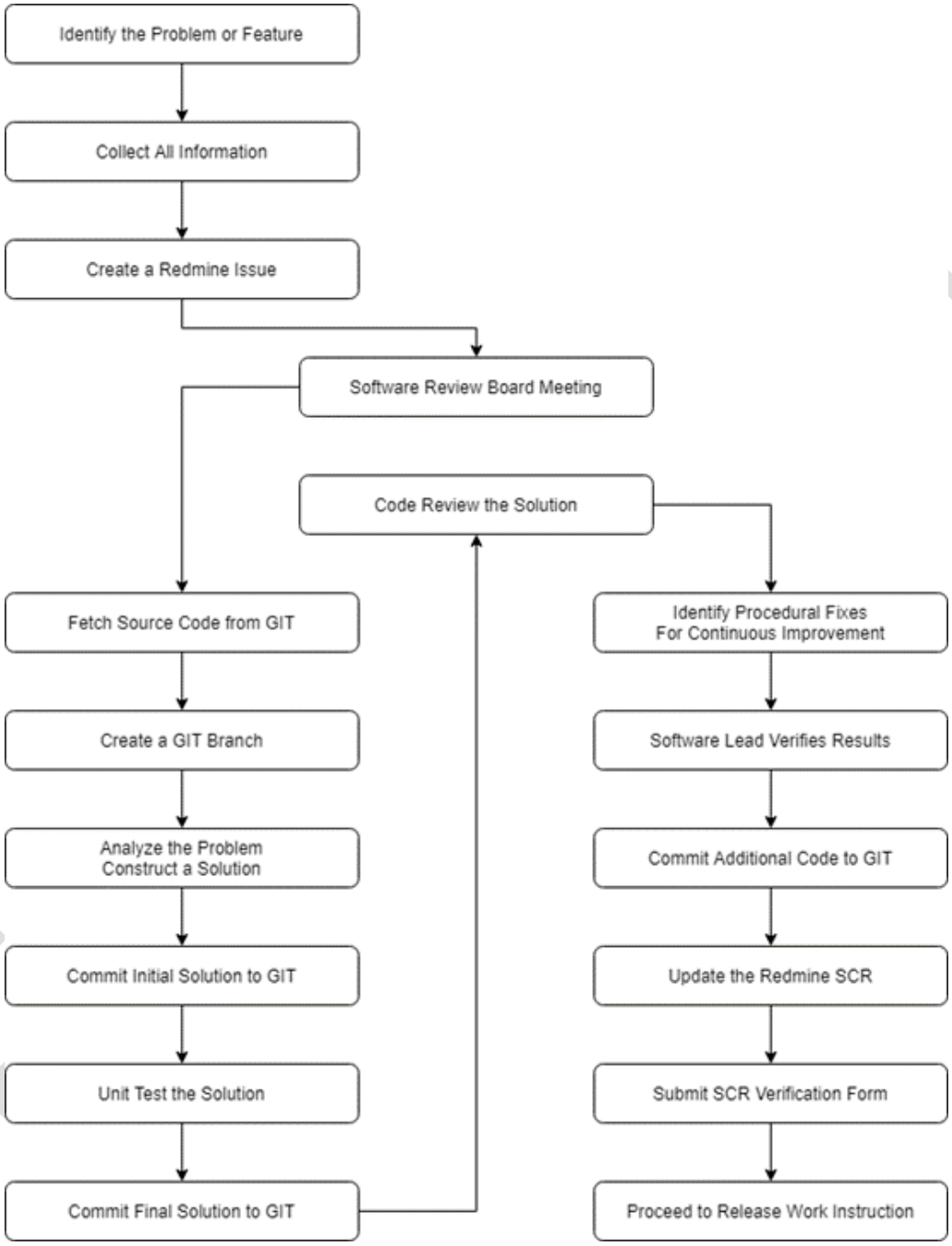

Figure 2: Software Maintenance Process

7.6 SOFTWARE PRODUCT READINESS REVIEW

This section describes the work instructions for the software product readiness review **PRR** for regulated flight software.

The release procedure and product readiness review depend highly on the specific product and development program. It is up to the program manager whether this specific work instruction applies to a particular software development effort. This procedure serves as an example of an appropriate release procedure for regulated flight software such as the Sagrad **AFTU** and **IAFTU**.

Please refer to the following document to locate specific details regarding the release procedure and product readiness review of a specific software product:

- QOP 8.3 Product Design and Development Rev G 12-13-2022

7.6.1 REGULATED FLIGHT SOFTWARE PRODUCT READINESS REVIEW

This procedure aims to provide a template for specific projects that need an internal validation step before releasing software to validate its adherence to the development process before delivery.

The **Flight Software Product Readiness Review** is a series of three meetings covering one of the three critical aspects of process validation before a major software release.

The end goal is the final publication of the **Master Validation Checklist** (MVC), associated with a release. This stringent procedure ensures that a lack of detail in any software release will be quickly located internally before it becomes externally visible.

Typically, this review series is coordinated the week of the system test or **run for record** so that delivery of the product and process align.

The three review phases are as follows.

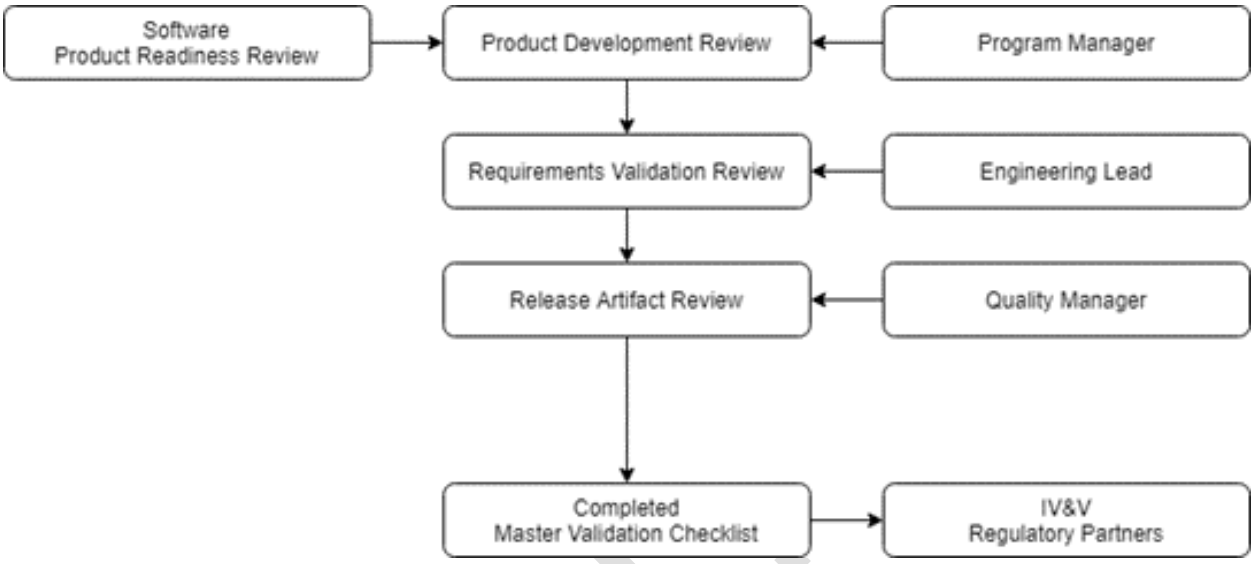

Figure 3: Software Product Readiness Review

7.6.1.1 PRODUCT DEVELOPMENT REVIEW

The first review meeting is the **Product Development Review**. In this review, the program manager briefly presents evidence that the following product phases have been completed. As a result, the quality manager checks these items on the **Master Validation Checklist**. It is acceptable if certain phases are missing if a reason is given and documented on the **MVC** by the quality manager.

- Statement of Work (SOW)
- Program Plan and Schedule
- Software Design Specification Document
- Preliminary Design Review Output and Issues
- Critical Design Review Output and Issues
- Software Implementation Review Output
- Production Readiness Review
- Acceptance Test Plan

- Quality/Qualification Test Procedure
- Technical Readiness Review (TRR) Output
- Quality Inspection Package Review
- Completed of User Manual

This review covers program management steps and design steps. Requirements validation and traceability are not presented until the following review. The engineering lead should also be present at this meeting to validate the final copy of the design documents.

7.6.1.2 REQUIREMENTS VALIDATION REVIEW

The **Engineering Lead** presents this review with the help of the **Configuration Manager**. The primary purpose is to perform a final internal validation of all requirements traceability before an external inspection by a regulatory partner such as **IV&V**. The initial regulatory requirement must be presented along with the software requirements specification, and it must be demonstrated that all requirements are traceable to either a system test or a manual validation inspection. Any missing elements or issues are discussed.

This is followed by a presentation of the system test plan and system test results, for instance, the **run for record**, and a brief description of how requirements are traced to each test. Any missing requirements are discussed. Unit tests and static analysis results are optional. The quality manager checks off all these elements on the **Master Validation Checklist** if they pass the group review.

The following items are presented.

- Present Regulatory Requirements such as RCC319-19T
- Software Requirements Specification
- Explain Tailoring of Regulatory Requirements
- Present System Test Plan
- Present Proof of Traceability Requirements to System Test Plan
- Software Code Review Results
- Software Static Analysis Results

- Software Unit Test Results
- Manual Validation Code Review Results
- Hazard Analysis Documentation
- Software System Test and Run for Record Results

7.6.1.3 RELEASE ARTIFACT REVIEW

During the release artifact review, the quality officer provides a snapshot of all documentation submitted as proof of the required process elements for the release. This is done in a group setting, with the opportunity to request missing elements before the release.

This includes the following:

- Location of the Master Development Folder
- GIT Tag of both the Software Source and the Master Development Folder
- Location of final copies of Software Requirements Specification and Design Specification
- Location of the final System Test Results or Run for Record Output
- Hazard Analysis Spreadsheet and Results
- Any Open SCR Trouble Reports
- A list of all items being submitted to regulatory review or IV&V
- Proof that everything is checked into source control (GIT)
- Presentation of the Master Validation Checklist and any missing items

The **GIT** tags and **Network** location of the **Software Executable** and **Master Validation Checklist** are entered into the **product configuration** sheet.

- SG932-0206 AFTU Configuration Data Report Rev F 7-20-2022.docx

Once the **Master Validation Checklist** is presented to the group for final internal approval, it is signed by the quality manager, program manager, and engineering lead. Then, the software is pronounced ready for delivery.

8 TABLE OF TABLES AND FIGURES

8.1 TABLE OF FIGURES

8.2 TABLE OF TABLES