*CAR PURCHASING PROJECT*

Verification & Validation Plan

Author: Hagar Mohamed

Creation Date: 1/5/2019

Version: 1.0

**Revision History**

Provide information on how the development and distribution of the Independent Verification and Validation Plan is controlled and tracked.

|  |  |  |  |
| --- | --- | --- | --- |
| Version No. | Date | Author | Revision Description |
| *1.0* | *1/5/2019* | *Hagar Mohamed* | *Documentation Creation* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[1. Approach 1](#_Toc7761517)

[1.1 Scope 1](#_Toc7761518)

[1.2 Objectives 1](#_Toc7761519)

[2. Life-Cycle Verification and Validation 2](#_Toc7761520)

[2.1 Requirement Phase 2](#_Toc7761521)

[2.1.1 Unambiguity 2](#_Toc7761522)

[2.1.2 Completeness 2](#_Toc7761523)

[2.1.3 Consistency 2](#_Toc7761524)

[2.1.4 Modifiability 2](#_Toc7761525)

[2.1.5 Traceability 2](#_Toc7761526)

[2.2 Architecture Phase 3](#_Toc7761527)

[2.3 Design Phase 3](#_Toc7761528)

[2.3.1 Requirements Traceability 3](#_Toc7761529)

[2.4 Development Phase 4](#_Toc7761530)

[2.4.1 Proactive Techniques 4](#_Toc7761531)

[2.5 Testing Phase 4](#_Toc7761532)

[2.5.1 Testing Plan 4](#_Toc7761533)

[2.5.2 Test design 5](#_Toc7761534)

[2.5.3 Test Cases 5](#_Toc7761535)

[3. References 7](#_Toc7761536)

# Approach

## Scope

The Software Verification and Validation Plan is produced using the IEEE Standard for Software Verification and Validation Plans (1012-1986) as a model. Software V&V employs review, analysis, and testing techniques to determine whether a software product and its intermediate deliverables comply with requirements. These requirements include both business functional capabilities and quality attributes Verification of requirement against defined specifications.

## Objectives

The objectives of the V&V effort are to find defects and to determine if required functions and attributes are built into the software system. V&V activities are designed to support:

1. Verification that the products of each software life cycle phase:

* Comply with previous life cycle phase requirements and products for correctness, completeness, consistency, and accuracy.
* Satisfy the standards, policies, practices, procedures, and conventions of the phase.
* Establish the proper basis for initiating the next life cycle phase.

1. Validate that the completed end product complies with established software and system requirements

**1.3 Activities of Verification and Validation**

* *Verification of design against defined specifications.*
* *Code reviews – Systematic examination of the product's source code*
* *Inspections – Peer review of work products and documentation*
* *Unit testing – Validates that individual units of product are working as designed*
* *Integration testing – Units of product are combined and tested as a group*
* *Function testing - Involves validating product functionality against defined requirements*
* *System testing – Testing of both hardware and software on a completely integrated system*
* *User acceptance testing – Black-box testing of product functionality to obtain release acceptance*
* *Determining the types and levels of product integrity to be verified and validated*
* *lanning and scheduling of IV&V activities considering the project management plan and schedule*
* *Consulting with stakeholders to assess their involvement regarding system functionality and the system's ability to meet their needs*
* *Reviewing and providing recommendations to improve both the management and technical aspects of the project including evaluating project progresses, resources, schedules, and reporting*
* *Reviewing and analyzing project management and software development activities and performance*
* *Reviewing traceability of product functions to original requirements*
* *Documenting IV&V activities and assessment results in the form of task reports, activity summary reports, anomaly reports, test documents, and eventually a final IV&V summary report.*

# Life-Cycle Verification and Validation

## Requirement Phase

To verify and validate our requirements, proactive and reactive techniques are chosen. These techniques thoroughly inspect the software requirements document for specific parameters. The parameters include unambiguity, completeness, verifiability, consistency, modifiability, traceability, and usability.

The primary objective of testing is to verify that the software we are developing is meeting client’s requirement or not. To make sure that we are on the right track we must examine our requirements (as we understand today) against prior documents. All of these requirements must be clear, concise, consistent, unambiguous, and testable.

### Unambiguity

All requirements must be examined closely to ensure no ambiguity exists in their specification. The developers must ensure that their interpretation of the requirements is free of ambiguity. This can be achieved through a series of document inspections. The inspector may include a moderator, developers, readers, inspectors, recorders, and client. In addition to this, we also plan to conduct client walkthrough sessions to ensure that everyone clearly understands requirements.

### Completeness

Even though it is often not possible to develop complete requirements during requirements phase, the last iteration should provide (at the very least) a complete and comprehensive requirements document. Here again, we will use a series of document reviews and client walkthroughs for transition between each of the phases to achieve this goal.

### Consistency

To ensure that the requirements gathered are consistent, we will periodically (throughout the software lifecycle) conduct a series of document reviews, especially during transition periods between phases. Each requirement will be compared against the previous and future versions to maximize consistency.

### Modifiability

Each requirement, including the diagrams, must be documented in a way that makes them easily modifiable. To achieve this, each requirement and diagram will be assigned a version number for easy revision. In addition, all deliverables are organized in templates to allow for convenient modifications.

### Traceability

The requirements traceability management involves adding, deleting, and changing requirements and their attributes. We will also organize and create views of different requirement types. Each requirement will be uniquely named. Any changes in the list of the requirements will be done with regards to the names.

## Architecture Phase

To validate our architecture we will use scenario-base method. We will build scenarios which illustrate the kind of activities and quality attributes that the system must support, for example, scenarios with respect to usability, scalability, and modifiability. Then we will describe candidate architectures, classify scenarios, and make overall evaluation.

To verify the architecture of the system during the phase the software engineer will follow organizational technique. This technique have to implemented from the beginning the phase. The following steps are determined:

1. We will develop an architecture design based on our SRS which was created in requirements phase.
2. At the end of each iteration (architecture cycle), we will inspect our architecture to validate it against requirements.
3. At the end of the architecture phase, we will document system architecture.
4. Once the document is created, we will review the document to make sure that the architecture documented is consistent with software requirements.
5. Inspections and review may lead to changes in the requirements—we will track these changes using version control.
6. We will use version control to keep track of changes in our architecture document.

## Design Phase

To verify our design phase, the software engineer will perform a formal inspection on the detailed design document. This will include inspecting the document for its conformity to applicable standards and inspecting the traceability of design elements to documented requirements. Then, quality attribute trade-offs, sensitivity points, and risks of the architectural style and possible alternatives will be considered.

In addition to using these techniques (tasks, documentation and tracking of important changes) it is also necessary to inspect the verification and validation process for effectiveness and possible improvements.

### Requirements Traceability

The goals of this review are to:

1. Ensure that the design fully addresses all the requirements including both functional and non-functional requirements.
2. Ensure that all the design elements are traceable to specific requirements i.e. the design should not have more than the requirements specified in the requirements document.
3. Ensure that the design is feasible for the developers. There should be no confusion or ambiguity. Ensure that we are building the software that our client requires.

A traceability matrix will be used to formally review the traceability of requirements to design elements in the system. The software engineer will analyze the system and populate the fields of the table such that a visual inspection of the table will clearly indicate whether all requirements have been fulfilled.

## Development Phase

The techniques outlined in this section are for the development phase of the lifecycle and are separated into two sections: Proactive and Reactive techniques. Proactive techniques refer to standards and protocol that will be followed by the developer in order to ensure defect avoidance. Reactive techniques refer to defect tracking and repair procedures that will be followed by the developer in order to repair errors. In addition to these two development focused V&V techniques, effectiveness assessment will also be performed.

### Proactive Techniques

#### Code Review

The developer will adhere to a strict coding standard to improve readability and modifiability of the code. After an iteration of development, a formal code inspection will be performed.

#### 2.4.2 Reactive Techniques

Defect Tracking

An output from the testing phase will be a list of defects with the last iteration produced by the developer. The developer will be responsible for maintaining this defect list as it strives to

## Testing Phase

### Testing Plan

The goal of this test plan is to ensure that the system’s functionality is as required and assists the QA in testing and verification of the software.

The test process is defined in terms of following phases:

* Test planning

We plan to debug during coding phase, unit testing for verification detailed design, integration testing for verification architecture design, and system testing for validation system (the system meets the requirements).

* Test design

We plan to use Use-Cases technique to design test cases for every test stage.

* Test execution

We plan to execute tests after each development phase using use cases manually.

* Test improvement

We plan to improve our test cases, if they would find small amount of defect or would not find defects at all.

### Test design

For managing the risk of releasing system, whose quality is unacceptable, the software engineer design set of tests, which follow one after another. Each test validates the system accordingly with its development phase, for example

1. Unit Test validates Detailed Design.
2. Integration Test validates Architecture Design.
3. Functional Test validates Software Requirements.
4. Acceptance (System) Test validates System Requirements.

### Test Cases

The test cases will be generated based on the Software Requirement Specification and the Configuration like that.

#### Functional Features

|  |  |
| --- | --- |
| **Test Case id #** |  |
| **Title** |  |
| **Description** |  |
| **Test data** |  |
| **Procedure** |  |
| **Test Steps** |  |
| **Expected Output** |  |
| **Related Requirement** |  |
| **Status** |  |
| **Author** |  |

Non-Functional Features

* **Performance**

Although the client doesn’t have strong requirements on performance, the execution time is expected to be bounded in the worst case.

* **Performance Scenario 1**

|  |  |
| --- | --- |
| **Environment** | Normal execution |
| **Stimulus** | Users run the application with valid input files. |
| **Response** | The application will display the rule validation result and save it on the log file within 60 seconds. |

# References

|  |  |
| --- | --- |
| [IEEE1] | IEEE Standard for Software Verification and Validation Plans (1012-1986) |
| [IEEE2] | IEEE Guide for Software Verification and Validation Plans (1059-1993) |
| [CMMI1.1] | Capability Maturity Model Integration, Version 1.1 (CMU/SEI-2002-TR-012) |
| [BASS 01] | Len Bass, Bonnie E. John, and Jesse Kates , Achieving Usability Through Software Architecture, CMU/SEI-2001-TR-005, March 2001 |