PROCESS

Requirement Development & Management

Code	**03e-QT/PM/HDCV/FSOFT**
Version	**1/3**
Effective Date	**20/11/2011**

TABLE OF CONTENT

1 INTRODUCTION

1.1 Purpose

The document introduces Requirement Development & Management Process aiming to provide an understanding that Requirement Development & Management Process

- Is not a discrete front-end activity of the software life cycle, but rather a process initiated at the beginning of a project and continuing to be refined throughout the life cycle
- Identifies software requirements as configuration items, and manages them using the same software configuration management practices as other products of the software life cycle processes
- Needs to be adapted to the organization and project context

In particular, the topic is concerned with how the activities of requirement elicitation, analysis, specification, and validation are configured for different types of projects and constraints.

The objectives of Requirement Development & Management Process are:

- To ensure that requirements for the software product are defined and understood.
- To establish and maintain agreement on the requirements with the requestor and affected groups (external customer, marketing, or another internal organization.)
- To ensure that the requirements are met.
- Requirements are documented and controlled to establish a basis for software development and project management use.

Changes to requirements are documented and controlled to ensure that plans, deliverables, and activities are consistent with requirements.

1.2 Application Scope

This process is applied at the entire FPT Software Company Limited.

1.3 Definitions

Abbreviations	Description
URD	User Requirement Document
SRS	Software Requirement Specification
BA	Business Analyst

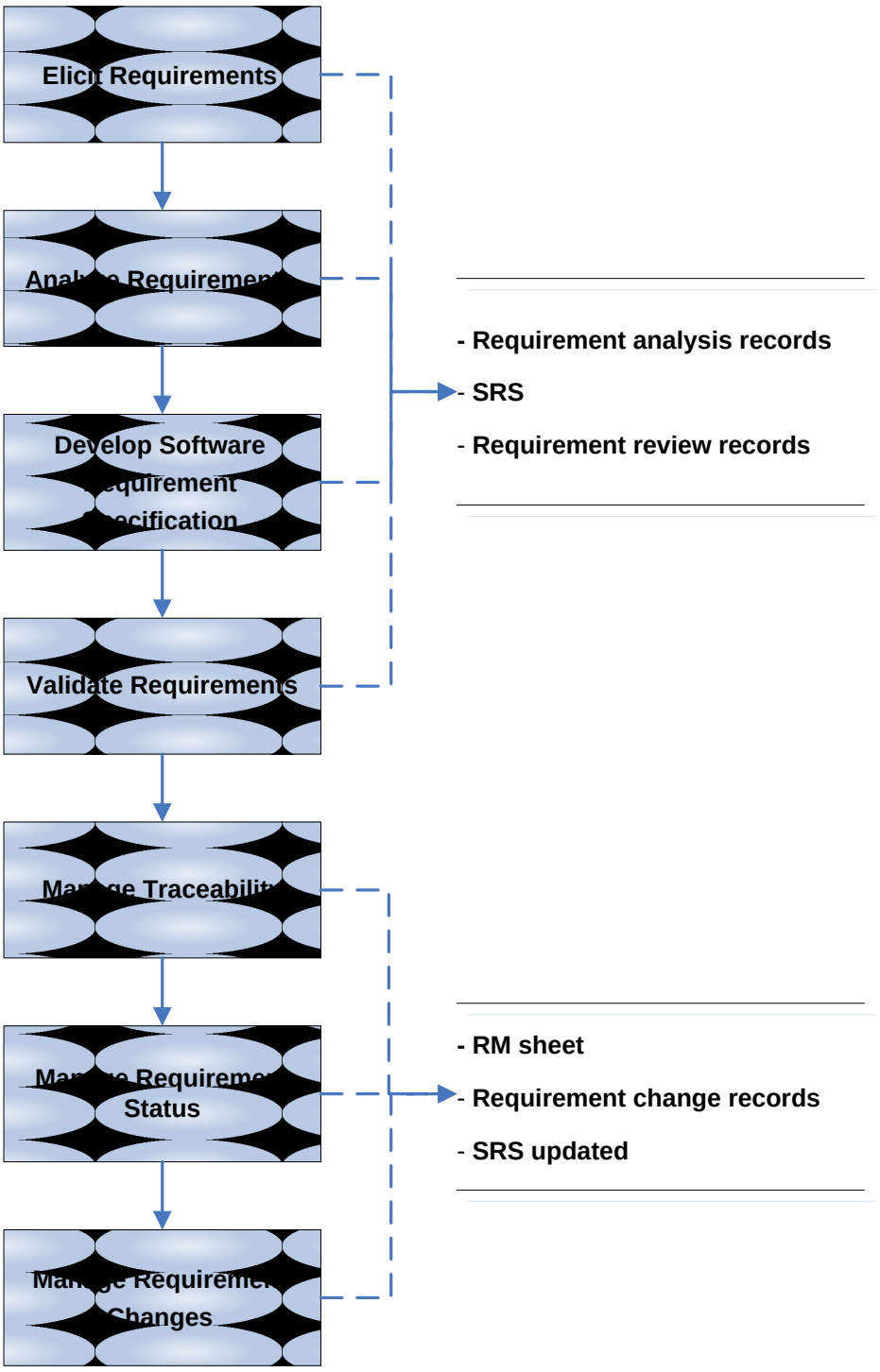
Abbreviations	Description
Requirement	is something wanted or needed. Hence, requirements on particular system/application are typically a complex combination of requirements from different people at different levels of an organization and from the environment in which the software will operate. Requirements are classified as
Requirement Specification	is a description of how a system should behave or a description of system properties or attributes. It can alternatively be a statement of "what" an application is expected to do
User Requirement	It is 1st sub-category of requirement. User Requirements address what users need to do their jobs. These requirements are implementation independent.
Process Requirement	It is 2nd sub-category of requirement. Process Requirements are constraints on the development of the product. Some product requirements generate implicit process requirements. The choice of verification technique is one example. Process requirements are imposed directly by the organization, customer or by the project itself, or a third party such as a safety regulator
Product Requirement	It is 3rd sub-category of requirement. Product Requirement s are requirements on product to be developed. These requirements specify a condition or capability that must be met or possessed by a system or its component(s). These include functional and non-functional requirements. Product requirement are developed to directly or indirectly satisfy user requirements.
Functional Requirements	Describes the function that the product is to execute; these include inputs, outputs, calculations, external interfaces, communications, and special management information needs. They are sometimes known as capabilities or statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
Non-functional Requirement	Is the ones that act to constrain the solution. Non-functional requirements are also known as attribute or quality requirements of the product. These requirements include areas such as performance, and design and implementation constraints.

1.4 Related Documents

No	Code	Name of documents
1	11e-QT/PM/HDCV/FSOFT	Process_Software Project Management
2	00a-CV/PM/HDCV/FSOFT	Job_FSOFT RADIO for Projects
3	06be-BM/PM/HDCV/FSOFT	Template_Software Requirement Specification
4	06ce-BM/PM/HDCV/FSOFT	Template_Use Case Specification
5	49e-BM/PM/HDCV/FSOFT	Template_Q&A Management sheet
6	03e-CL/PM/HDCV/FSOFT	Checklist_SRS Review
7	04e-CL/PM/HDCV/FSOFT	Checklist_Use Case Review
8	07e-CL/PM/HDCV/FSOFT	Checklist_GUI Review
9	06de-BM/PM/HDCV/FSOFT	Template_Requirement Management Sheet
10		Tool_Fsoft Insight
11		Process_Software contract management
12	14e-QT/PM/HDCV/FSOFT	Process_Design
13	02e-QT/PM/HDCV/FSOFT	Process_Configuration Management
14	05e-QT/PM/HDCV/FSOFT	Process_Test

2 PROCESS

2.1 Work flow



2.2 Process Description

No	Entry Criteria	Activity	Exit Criteria
1	- Project is opened and PM is appointed	Requirement Development - Elicit Requirements - Analyze Requirements - Develop Software Requirement Specification - Validate Requirements	- Requirements are developed and analyzed
2	- Project is opened and PM is appointed - Requirements are provided	Requirement Management - Manage Requirement Traceability - Manage Requirement Status - Manage Requirement Changes	- RM sheet is updated with requirement status - Changes to requirements are approved/declined - Changes are communicated to relevant people and implemented as per planned

2.3 Roles and Responsibility

The people who participate in the requirements process will vary across projects, but always include:

Requirement Providers: Typical examples of requirements providers include (but are not restricted to)

- *Users*: This group comprises those who will operate the software. It is often a heterogeneous group comprising people with different roles and requirements.
- *Customers*: This group comprises those who have commissioned the software or who represent the software's target market.

***Users* and *Customers* are groups of people who will be directly or indirectly impacted by the products.**

- *Software engineers*: These individuals have a legitimate interest in profiting from developing the software by, for example, reusing components in other products. If, in this scenario, a customer of a particular product has specific requirements which compromise the potential for component reuse, the software engineers must carefully weigh their own stake against those of the customer.

- *Market analysts*: A mass-market product will not have a commissioning customer, so marketing people are often needed to establish what the market needs and to act as proxy customers.
- *Regulators*: Many application domains such as banking and public transport are regulated. Software in these domains must comply with the requirements of the regulatory authorities.

Depending on project characteristics and customer's organization, a single point of contact (SPOC) might be identified for customers in order to gather provided requirements through single channel of interface. For internal projects, the requirement providers are senior managers or marketing department of the organization. The requirement providers normally are specified in the project plan but it can also be specified in detail against each requirement and requirement change in the requirement management.

Business Analyst (BA): the person in charge of requirement analysis and management in projects, that role is assigned by PM. If no specific assignment, PM will handle that job by default.

Change Control Board (CCB): is a committee that makes decisions regarding whether or not proposed changes to a project should be implemented. The authority of the change control board may vary from project to project, but decisions reached by the CCB are often accepted as final and binding. CCB s constituted of project team, customer and relevant stakeholders or their representatives, include (but not limited to):

- *Customer side:* Customer's PM.
- *Project Team:* PM, PTL, CC, PM chair the CCB internally project.

Particularly, the following changes are requested to be approved by correspondent OM/SM (depending on project rank):

- Changes to project scope
- Changes in target of project QPPO
- Changes in delivery plan of project deliverables
- Change to project resource for key roles (SM, PM), or the resource that have commitment about the HR, such as Time-material based projects

2.4 Work Product

- **URD**: document which compose all business requirements formulated by customer, business rules and other constrains to software.
- **SRS**: document which serve as common reference point of the software requirements for customer, developer, tester and Project Manager
- **Requirement Prototype (optional)**: as part of SRS.

- **Change Request**: document which compose changes to requirements
- **RM Sheet**: excel sheet, used to track the status, relationship and change of requirements during the whole project.

2.5 Metrics

Refer to Guideline_Software Metrics

2.6 Record

No	Name of Records	Storage Modality	Duration (year(s))	Important Level	Location
1	URD	Soft	3 years	Medium	Project Repository
2	SRS	Soft	3 years	Medium	Project Repository
3	Requirement Prototype	Soft	3 years	Medium	Project Repository
4	RM sheet	Soft	3 years	Low	Project Repository
5	Change Request	Soft	3 years	Medium	Project Repository
6	SRS Review Report	Soft	3 years	Low	Project Repository
7	URD Review Report	Soft	3 years	Low	Project Repository
8	Customer Approval Records	Soft	3 years	Low	Project Repository
9	Requirement Q&A Records	Soft	3 years	Low	Project Repository

3 REQUIREMENT DEVELOPMENT PROCEDURE

3.1 Purpose

Simply, **Requirement Development** aims to identify and document the customer requirements for a product/service. This is the process of understanding the customer needs and expectation from a product/service. It covers the complex task of eliciting and documenting the requirement of all requirement providers, modeling and analyzing these requirements and specifying them as a basis for product/service development and project management use

3.2 Trigger

Requirement Development is in scope of projects

3.3 Inputs

- Contract/External Work Order

3.4 Steps

3.4.1 Elicit Requirements

Requirement elicitation activities begin by understanding the organization's "business requirements". This leads to determining project scope and purpose, the background leading to the decision to develop a new or modified system. This initial step helps in identifying how new system integrates with the business processes, how it fits into the larger picture and what the scope and limitations will be. Project scope & purpose are often vaguely formulated, PM and BA need to pay particular attention to assessing the value (relative to priority) and cost.

The next step is to gather the stated requirement from various sources. To do it, the project shall identify requirement providers – who is likely to be affected by the introduction of the system, in which "users" and "customers" are mandatory required. By defining in concrete terms that the customer and intended user are, the BA knows in advance where he has to look for answers. If lacking consideration of everyone who is likely to be requirement provider, there is a great likelihood of missing some critical requirements.

Once the requirements providers have been identified, projects can start eliciting requirements from them. It is a very difficult area and the software engineer needs to be sensitized to the fact that (for example) users may have difficulty describing their tasks, may leave important information unstated, or may be unwilling or unable to cooperate. It is particularly important to

understand that elicitation is not a passive activity, and that, projects have to work hard to elicit the right information.

3.4.2 Analyze Requirements

Once all stakeholder requirements have been gathered, Requirement analysis is done with the purpose

- Detect and resolve conflicts between requirements
- Discover the bounds of the software and how it must interact with its environment
- Elaborate system requirements to derive software requirements

To analyze requirements, projects to

a) Build System Model

The developers should construct an 'Implementation Independent Model' of what is required by the user. This is called a Logical Model that is used to produce software requirements. This model may be constructed by different methods described in following:

Functional Decomposition

Functional Decomposition is the traditional method of analysis. The emphasis is on ‘what’ functionality must be available, not ‘how’ it is to be implemented. The functional breakdown is constructed top-down, producing a set of function, sub-functions and functional interfaces.

A logical model of good quality by this method should satisfy the following requisites:

- The number of interfaces should be minimal, so that design components with weak coupling are not derived easily
- Functions should have a single definite purpose. Function names should have a declarative structure (E.g.: 'Validate Tele commands') and say 'what is to be done' rather than 'how to do it'
- Function level entries should be appropriate (E.g.: 'Calculate Checksum' should not appear at the same level as 'Verify Tele commands')
- Each function should not be decomposed into more than seven sub-functions
- The model should omit implementation information like file, record, task, module, etc.
- Performance attributes (speed, response time, etc.) of each function should be stated
- Critical functions should be identified

Object Oriented Analysis

Object Oriented Analysis (OOA) is the name for a class of methods that analyze a problem by studying the objects in the problem domain.

An object is ‘an abstraction of something in the domain of a problem or its implementation, reflecting the capability of the system to keep information about it, interact with it or both; an encapsulation of attributes values and their exclusive services. A class describes a set of objects with common attributes and services. An object is an ‘instance’ of a class and the act of creating an object is called ‘instantiation’

Classes can be decomposed into sub-classes. Subclasses share family of characteristics, and OOA provides for this by permitting subclasses to inherit operations and attributes from their parents.

OOA differs from structured analysis by:

Building an object model first, instead of functional model (i.e. hierarchy of dataflow diagrams)

Integrating object, attributes and operations, instead of separating them from the data model and the functional model

Conceptual Modeling

Conceptual models comprise models of entities from the problem domain configured to reflect their real-world relationships and dependencies.

Several kinds of models can be developed. These include data and control flows, state models, event traces, user interactions, object models, data models, and many others. The factors that influence the choice of model include

- The nature of the problem. Some types of software demand that certain aspects be analyzed particularly rigorously. For example, control flow and state models are likely to be more important for real-time software than for management information software, while it would usually be the opposite for data models.
- The expertise of the software engineer. It is often more productive to adopt a modeling notation or method with which the software engineer has experience.
- The process requirements of the customer. Customers may impose their favored notation or method, or prohibit any with which they are unfamiliar. This factor can conflict with the previous factor.
- The availability of methods and tools. Notations or methods which are poorly supported by training and tools may not achieve widespread acceptance even if they are suited to particular types of problems.

Note that, in almost all cases, it is useful to start by building a model of the software context. The software context provides a connection between the intended software and its external environment. This is crucial to understanding the software’s context in its operational environment and to identifying its interfaces with the environment.

The issue of modeling is tightly coupled with that of methods. For practical purposes, a method is a notation (or set of notations) supported by a process which guides the application of the notations. There is little empirical evidence to support claims for the superiority of one notation

over another. However, the widespread acceptance of a particular method or notation can lead to beneficial industry-wide pooling of skills and knowledge. This is currently the situation with the UML (Unified Modeling Language).

Formal modeling using notations based on discrete mathematics, and which are traceable to logical reasoning, have made an impact in some specialized domains. These may be imposed by customers or standards or may offer compelling advantages to the analysis of certain critical functions or components.

Rapid prototyping

A prototype is an executable model of selected aspects of a proposed system. If the requirements are not clear, or suspected to be incomplete, it can be useful to develop a prototype based on tentative requirements to explore what the software requirements really are. Prototypes can help define user interface requirements.

Rapid prototyping is 'the process of quickly building and evaluating a series of prototypes'. Specification and development can be iterative

A scenario is a sequence of events that might occur in the use of the product, which is used to make explicit some of the needs of the stakeholders. In contrast, an operational concept for a product usually depends on both the design solution and the scenario. For example, the operational concept for a wireless-based communications product is quite different from one based on landlines (wired). Since the alternative solutions have not usually been defined when preparing the initial operational concepts (including functionality, support, maintenance, and disposal), conceptual solutions are developed for use when analyzing the requirements. The operational concepts are refined as solution decisions are made and lower level detailed requirements are developed.

Just as a design decision for a product may become a requirement for product components; the operational concept may become the scenarios (requirements) for product components.

The scenarios may include operational sequences, provided those sequences are an expression of customer requirements rather than operational concepts.

b) Classify Requirements

Requirements are classified as Process Requirements & Product Requirements, where Product Requirements are divided into

- Functional Requirements
- Non-functional Requirements: those could be further categorized to Performance, Interface, Usability, Quality, Reliability, Design constrains, Operational, Documentation, Security

3.4.3 Develop Software Requirement Specification

Requirements, once elicited, modeled and analyzed should be documented in clear, unambiguous terms. Developing Software Requirement Specification, focus of the project

moves from the broad statement of user needs, goals and objectives, target markets and features of the system to the details of how these features are going to be implemented in the solution. What we need now is a collection, or package, of work products that says specifically, “Here is what the system has to do to deliver those features.” That is what we refer to as Software Requirement Specification (SRS)

Well-designed, clearly documented SRS is vital and serves as a:

- Base for validating the stated requirements and resolving stakeholder conflicts, if any
- Contract between the client and development team
- Basis for systems design for the development team
- Bench-mark for project managers for planning project development lifecycle and goals
- Source for formulating test plans for QA and testing teams
- Resource for requirements management and requirements tracing
- Basis for evolving requirements over the project life span

SRS involves scoping the requirements so that it meets the customer’s vision. It is a document that lists out stakeholders’ needs and communicates these to the software engineer that will design and build the system. The challenge of a well-written SRS is to clearly communicate to both these groups and all the sub-groups within.

To overcome this, SRS may be documented separately as

- **User Requirements** - written in clear, precise language with plain text and use cases, for the benefit of the customer and end-user
- **Software Requirements** - expressed as a programming or mathematical model, addressing the Application Development Team and QA and Testing Team.

SRS serves as a starting point for software, hardware and database design. It describes the function (Functional and Non-Functional specifications) of the system, performance of the system and the operational and user-interface constraints that will govern system development.

In case the project does not involve production of the URD, but does involve writing the SRS, the activities in the user requirement analysis should be performed to capture the user's requirements.

Characteristic of a Good SRS

Quality of requirement affects work performed in subsequent phases of the system lifecycle. Poor quality requirements:

- Increase cost and schedule: effort is spent during design and implementation trying to figure out what the requirements are
- Decrease product quality: poor requirements cause the wrong product to be delivered or de-scoping to meet schedule or cost constraints

- Increase maintenance effort: lack of traceability increase the effort to identify where changes are required, especially as knowledgeable personnel leave
- Create disputes with the customer/client: ambiguity causes differences in expectations and contractual issues
- Are a major cause of project failure: all of above

Requirements are considered as GOOD if they are

Necessary	If the system can meet prioritized real needs without the requirement. It is not necessary
Feasible	The requirement is doable and can be accomplished within available cost and schedule
Correct	The facts related to the requirement are accurate and it is technically and legally possible
Concise	Requirements are stated simply, used word “Shall”, not used **if**, **when**, **but**, **except**, **unless** and **although** and do not include speculative or general terms such as **usually**, **generally**, **often**, **normally** and **typically**
Unambiguous	The requirement can be interpreted in only one way Use unique terms/names throughout the document. Define all terms and expand acronyms at the very beginning of the SRS. Use simple structured language to eliminate ambiguity.
Complete	All conditions under which the requirement applies are stated, and the requirement expresses a whole idea or statement.
Consistent	The requirement is not in conflict with other requirements. Requirements must use consistent terminology A glossary helps achieve consistent terminology. Even commonly understood terms can create headaches later if they are undefined because they can change meaning over time. That is why acronyms common to the users’ business shall be define in the glossary for newcomers to the project
Verifiable	Implementation of the requirement in the system can be proved
Traceable	The requirement can be traced to its source, and it can be tracked throughout the system (e.g., to the design, code, test and documentation).
Non-Redundant	The requirement is not a duplicate requirement
Associated with information	Associated information of requirements is necessary for managing the project as well as the requirements. For example, associating priority and

	WBS with requirements facilitates planning and scheduling, associating change history with the requirement facilitates change control, while other associated information is necessary to produce traceability and other requirements management reports.

Prepare verification and acceptance criteria: Verification requirements and Acceptance criteria specify the constraints on how the software is to be verified and accepted. These may include requirements for simulation, emulation, live tests with simulated inputs, live tests with real inputs and interfacing with the testing environment.

3.4.4 Validate Requirements

The requirements documents are subject to validation and verification procedures. The requirements must be validated to ensure that the software engineer has understood the requirements, and it is also important to verify that a requirements document conforms to organizational standards, and that it is understandable, consistent, and complete. Formal notations offer the important advantage of permitting the last two properties to be proven (in a restricted sense, at least). Different stakeholders, including representatives of the customer and developer, shall review the document(s).

It is normal to explicitly schedule one or more points in the requirements process where the requirements are validated. The aim is to pick up any problems before resources are committed to addressing the requirements. Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right software (that is, the software that the users expect).

Requirements Review is perhaps the most common mean of validation. A group of reviewers is assigned a brief to look for errors, mistaken assumptions, lack of clarity, and deviation from standard practice. Reviewers conducts the review is important (at least one representative of the customer should be included for a customer-driven project, for example), and it may help to provide guidance on what to look for in the form of checklists.

Reviews may be constituted on completion of URD, SRS, the baseline specification for a new release, or at any other step in the process.

Validation on requirement understanding is also very important in order to avoid requirement mis-understanding among project team members and between project team with the customer. Standard requirement understanding validation methods are: 1) requirement group-review meeting via tv meeting, tele conference, chat or face-to-face meeting; 2) requirement understanding is written down and get validated/confirmed with the requirement provider; 3) prototyping.

Prototyping is commonly a means for validating the software engineer's interpretation of the software requirements, as well as for eliciting new requirements. As with elicitation, there is a range of prototyping techniques and a number of points in the process where prototype

validation may be appropriate. The advantage of prototypes is that they can make it easier to interpret the software engineer's assumptions and, where needed, give useful feedback on why they are wrong. For example, the dynamic behavior of a user interface can be better understood through an animated prototype than through textual description or graphical models. There are also disadvantages, however. These include the danger of users' attention being distracted from the core underlying functionality by cosmetic issues or quality problems with the prototype. For this reason, several people recommend prototypes which avoid software, such as flip-chart-based mockups. Prototypes may be costly to develop. However, if they avoid the wastage of resources caused by trying to satisfy erroneous requirements, their cost can be more easily justified.

Model Validation is typically necessary to validate the quality of the models developed during analysis. For example, in object models, it is useful to perform a static analysis to verify that communication paths exist between objects which, in the stakeholders' domain, exchange data. If formal specification notations are used, it is possible to use formal reasoning to prove specification properties.

Acceptance Tests: An essential property of a software requirement is that it should be possible to validate that the finished product satisfies it. Requirements which cannot be validated are really just "wishes." An important task is therefore planning how to verify each requirement. In most cases, designing acceptance tests does this.

Identifying and designing acceptance tests may be difficult for non-functional requirements. To be validated, they must first be analyzed to the point where they can be expressed quantitatively.

3.5 Outputs

- URD
- SRS
- Requirement Prototype (optional)
- Requirement Q&A Records
- Review Reports of URD, SRS
- Records of URD, SRS Approval

4 REQUIREMENT MANAGEMENT PROCEDURE

4.1 Purpose

It is to organization and management of Requirements to ensure all requirements are satisfactorily implemented and accepted.

4.2 Trigger

- Project is opened and PM is appointed

4.3 Inputs

- Requirements are baselined

4.4 Steps

4.4.1 Manage Traceability

When executing software projects, keep the followings in mind

- User requirements form the basis of final validation tests.
- System requirements form the basis of design and tests.
- System requirements are inputs to project planning estimates.
- All requirements are traced from business requirement, met in design and tested
- Change to one requirement may have effect on all other items this requirement is linked with.

Traceability matrix is established and maintained to provide a traceability analysis or matrix which links requirements, design specifications, and validation. Traceability among these activities and documents is essential. This document acts as a map, providing the links necessary for determining where information is located.

- It demonstrates the relationship between design inputs and design outputs
- It ensures that design is based on predecessor, established requirements
- It helps ensure that design specifications are appropriately verified, that functional requirements are appropriately validated

Traceability across the Life Cycle

1. Risk Assessment (Initial and Ongoing Activities)
    1. Trace potential risks to their specific cause
    2. Trace identified mitigations to the risk
    3. Trace specific causes of software-related risk to their location in the software
2. Requirements Analysis and Specification
    1. Trace Software Requirements to User Requirements
    2. Trace Child Requirements to their Parent Requirement
    3. Trace Software Requirements to hardware, user, operator and software interface requirements
3. Design
    1. Trace Architectural Design to Software Requirements
    2. Trace Design Interfaces to hardware, user, operator and software interface requirements
    3. Trace Detailed Design to Architectural Design
4. Source Code Analysis
    1. Trace Source Code to Detailed Design Specifications
    2. Trace unit tests to Source Code and to Design Specifications
5. Integration Test
    1. Trace integration tests to Architectural Design
    2. IMPORTANT: Use Architectural Design to establish a rational approach to integration, to determine regression testing when changes are made
6. System Test
    1. Trace system tests to Software Requirement Specifications
    2. Use a variety of test types
    3. Design test cases to address concerns such as robustness, stress, security, recovery, usability, etc.
    4. Use traceability to assure that the necessary level of coverage is achieved

4.4.2 Manage Requirement Status

Refer to Template_Requirement Management Sheet

4.4.3 Manage Requirement Changes

During project planning, a PM decides which process is to be followed for handling change requests. The planned process is discussed with the customer so that both the customer and the project team are in agreement about how to manage changes.

Generally, the process specifies how the change requests will be made, when formal approvals are needed, and so on. When a request for a requirements change comes in, the requirements change management process must be executed.

Because change requests have cost implications, it is necessary to have a clear agreement on payment. Frequently, with customer approval, projects build a buffer into their estimates for

implementing change requests (typically a small percentage of the total project effort). Such a budget provision simplifies the administrative aspects of implementing approved change requests.

The commonly used change management process has the following steps:

- Log the changes in CR form.
- Perform an impact analysis on the work products.
- Estimate the effort needed for the change requests.
- Re-estimate the delivery schedule.
- Perform a cumulative cost impact analysis.
- Review the impact with senior management if thresholds are exceeded.
- Obtain customer sign-off.
- Rework work products through its life cycle

You maintain a change request log in Template_Requirement Management Sheet to keep track of the change requests. Each entry in the log contains a change request number, a brief description of the change, the effect of the change, the status of the change request, and key dates. You assess the effect of a change request by performing impact analysis. Impact analysis involves identifying work products that need to be changed and evaluating the quantum of change to each; reassessing the project's risks by revisiting the risk management plan; and evaluating the overall implications of the changes for the effort and schedule estimates. The outcome of the analysis is reviewed and approved by the customer. The change requests are incorporated in the Requirements Specification Document, usually as appendixes. Sometimes the relevant portions of the document are also modified to reflect the changes. Monitoring of approved change requests and ensuring their proper implementation are handled by the configuration management process.

A change might be classified as minor if the total effort involved in implementing it does not exceed a predetermined value—say, two-days. Minor changes typically become part of the project effort, utilizing the buffer in the planned estimate. Major changes usually have a larger impact on effort and schedule and must be formally approved by the client. Senior management gains visibility in the changes through status and milestone reporting.

Note that although the impact of a change request is specified, and approved, by using the template, the actual tracking of implementation of a change request is handled by the CM process.

One danger of requirement changes is that, even though each change is not large in itself, over the life of the project the cumulative impact of the changes is large. Hence, in addition to studying the impact of and tracking individual changes, you must monitor the cumulative impact of changes. For cumulative changes, Template_Requirement Management Sheet are used. To facilitate this analysis,

the logs are frequently maintained. From that file, you can immediately see the total cost of the requirement changes made so far. PMs sometimes plan some buffer for handling change requests, so as long as the cumulative effort for all change requests is less than this buffer, nothing special needs to be done. If the cumulative effort of all changes exceeds this buffer, however, further changes can have an adverse effect on total cost and scheduling. In this situation, the PM must revise the estimates and get them approved.

4.5 Outputs

- Change Requests
- Revisions of SRS
- RM Sheet

5 GUIDELINES

5.1 Requirement Elicitation Techniques

A number of techniques exist for doing requirement elicitation. These techniques can be used in combination.

5.1.1 Document Analysis

All effective requirements elicitation involves some level of document analysis such as business plans, market studies, contracts, requests for proposals, statements of work, existing guidelines, analyses of existing systems, and procedures. Improved requirements coverage results from identifying and consulting all likely sources of requirements. All project members assigned to this activity should study all related documents in depth. This ensures clear understanding of the process by key project staff, and will help them during later stages of development and testing.

5.1.2 Brainstorming

Brainstorming involves both idea generation and idea reduction. The goal of the former is to identify as many ideas as possible, while the latter ranks the ideas into those considered most useful by the group. Brainstorming is a powerful technique because the most creative or effective ideas often result from combining seemingly unrelated ideas. Also, this technique encourages original thinking and unusual ideas. Various voting techniques may be used to prioritize the ideas created. Besides preferred live brainstorming, web-based brainstorming may be viable alternative in some situations.

Rules for Brainstorming technique are:

- Do not allow criticism or debate
- Let your imagination soar
- Generate as many ideas as possible
- Mutate and combine ideas
- Idea Reduction
    - Pruning ideas that are not worthy for further discussion
    - Grouping of similar ideas into one super topic
- Prioritize the remaining ideas

5.1.3 Facilitated meeting

The purpose of these is to try to achieve a summative effect whereby a group of people can bring more insight into their software requirements than by working individually. They can brainstorm and refine ideas which may be difficult to bring to the surface using interviews. Another advantage is that conflicting requirements surface early on in a way that lets the stakeholders recognize where there is conflict. When it works well, this technique may result in a richer and more consistent set of requirements than might otherwise be achievable. However, meetings need to be handled carefully (hence the need for a facilitator) to prevent a situation from occurring where the critical abilities of the team are eroded by group loyalty, or the requirements reflecting the concerns of a few outspoken (and perhaps senior) people are favored to the detriment of others.

5.1.4 Interviews

Interviews are used to gather information. However, the predisposition, experience, understanding, and bias of the person being interviewed influence the information obtained. The use of context-free questions by the interviewer helps avoid prejudicing the response. A context-free question is a question that does not suggest a particular response. For example, who is the client for this system? What is the real reason for wanting to solve this problem? What environment is this product likely to encounter?

Then, it may be appropriate to search for undiscovered requirements by exploring solutions. Convergence on some common needs will initiate a “requirement repository” for use during the project. A questionnaire is no substitute for an interview.

5.1.5 Requirements Workshops

Requirements workshop is perhaps the most powerful technique for eliciting requirements. It gathers all **key** requirement providers together for a short but intensely focus period. The use of outside facilitator experienced can ensure the success of the workshop. Other advantages are often achieved -- participant commitment to the work products and project success, teamwork, resolution of political issues, and reaching consensus on a host of topics. Benefits of requirements workshops include the following:

- Workshop costs are often lower than are those for multiple interviews.
- They involve users and cut across organizational boundaries.
- They help to identify and prioritize needs and resolve contentious issues.
- When properly run, they help to manage user's expectations and attitude toward change

A special category of requirements workshop is a Joint Application Development (JAD) workshop. JAD is a method for developing requirements through which customers, user

representatives, and developers work together with a facilitator to produce a requirements specification that both sides support. This is an effective way to define user needs early.

Rules for Requirement workshop technique are:

- Prepare for the workshop:
    - Set an agenda before the workshop and publish it along with the other pre-workshop documentation 1 day prior to the meeting
    - Selling the workshop concepts to the target participants
    - Ensure the participation of the Right Requirement providers
    - Well prepared logistic: venue, tea-break/lunch, equipments: notebook, projector, network, etc
    - Warm-up material: project-specific information, out-of box thinking preparation, etc.
- During the workshop: Role of Facilitator
    - Start and stop the meeting on time
    - Establish and force the “rules” for the meeting
    - Introduce goals and agenda of the meeting
    - Manage the meeting and keep the team “on-track”. Try to stay on the agenda, but do not strictly obey it, especially if good discussion is going on.
    - Facilitate a process of decision and consensus making but avoid participating in the content
    - Make certain that ALL participants participate and have their input heard
    - Control disruptive or unproductive behavior

5.1.6 Prototyping

Prototyping is a technique for building a quick and rough version of a desired system or parts of that system to help developers, users and customer better understand system requirements. It is a valuable tool for clarifying “fuzzy” requirements, those that, although known or implied, are poorly defined and poorly understood. It serves as a communications mechanism to provide users with a context within which they can better understand interactions with the system. Prototyping sometimes gives an impression that developers are further along than is actually the case, giving users an overly optimistic impression of completion possibilities. Prototypes can be combined effectively with other approaches such as JAD and models. There is a wide range of prototyping techniques, from paper mock-ups of screen designs to beta-test versions of software products, and a strong overlap of their use for requirements elicitation and the use of prototypes for requirements validation.

5.1.7 Interfaces Analysis

Missing or incorrect interfaces are often a major cause of cost overruns and product failures. Identifying external interfaces early clarifies product scope, aids risk assessment, reduces

product development costs, and improves customer satisfaction. The steps of identifying, simplifying, controlling, documenting, communicating, and monitoring interfaces help to reduce the risk of problems related to interfaces.

5.1.8 Storyboards

Storyboards identify the players, explain what happen to them, and describe how it happens. Storyboards are a kind of paper prototyping. Customers, users, or developers start by drawing pictures of the screens, dialogs, toolbars, and other elements they believe the software should provide. The group continues to evolve these until real requirements and details are worked out and agreed upon. Storyboards are inexpensive and eliminate risks and higher costs of prototyping. Another related technique is storytelling: the writing of vignettes to envision new products and services based on perceived user needs and the possibilities offered by emerging technologies.

5.1.9 Use Cases

Like storyboards, Use Cases identify the who, what and how of system behavior. User cases describe the interactions between a user and a system, focus on what the system “does” for the user. It should be accompanied by a textual description and not be used in isolation of other requirements gathering techniques. Use cases should always be supplemented with quality attributes and other information such as interface characteristics. The user case model describes the totality of the systems functional behavior. Many developers believe that use cases and scenarios (descriptions of sequences of events) facilitate team communication. They provide a context for the requirements by expressing sequences of events and a common language for end users and the technical team.

Be cautioned that use cases alone do not provide enough information to enable development activities. Other requirements elicitation techniques should also be used in conjunction with use cases. It is recommends using operational concepts as a simple, cost-effective way to build a consensus among stakeholders and to address two large classes of requirements errors: omitted requirements and conflicting requirements. Operational concepts identify user interface issues early, provide opportunities for early validation, and form a foundation for testing scenarios in product verification.

5.1.10 Scenarios

It is a valuable mean for providing context to the elicitation of user requirements. They allow the software engineer to provide a framework for questions about user tasks by permitting “what if” and “how is this done” questions to be asked. The most common type of scenario is the use case.

5.1.11 Performance and Capacity Analysis

Stakeholders emphasized that concentrating on system functions and data resulted in a lack of attention to the total system requirements and in incomplete performance, capacity, and external interface requirements. Thus, it is vital to ensure that the requirements gathering process provides for all requirements (requirements coverage).

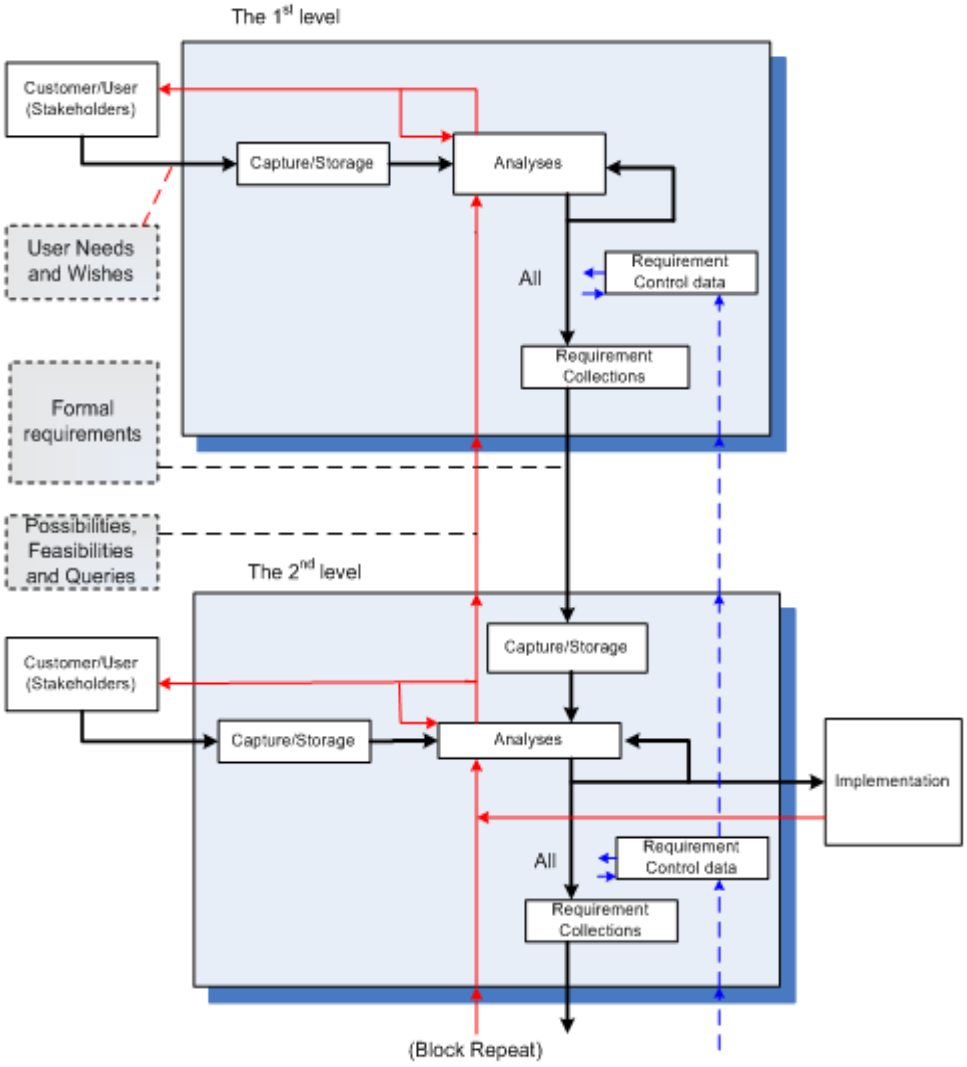
5.1.12 Role playing

Role playing allows the project team to experience the user's world from the user's perspective by immersing themselves in the environment and observing how users interact with their software and with each other. These techniques are relatively expensive, but they are instructive because they illustrate that many user tasks and business processes are too subtle and complex for their actors to describe easily.

Information is gathered from the various sources, the BA draws out from each of these groups what their requirements from the system are and what they expect the system to accomplish.

It will not be possible to perfectly satisfy the requirements of every stakeholder, and it is the software engineer's job to negotiate trade-offs which are both acceptable to the principal stakeholders and within budgetary, technical, regulatory, and other constraints. A prerequisite for this is that all the stakeholders be identified, the nature of their "stake" analyzed, and their requirements elicited.

Figure below is the representation of the requirement data flow where users needs/wishes evolve



The heavy, generally downward directed lines represent the requirement flow. It starts with the needs and desires of the top level customer/stakeholder and goes down through the first analyses to the first requirement collections. From there the flow goes down through the capture/storage and the second level analyses to the second level requirement collections. The first level requirement collections might correspond to user requirement document (URD) describing scenarios (both basic and alternatives) that users and stakeholders expect from the system, and the second to software requirement specifications (SRS).

The customers/stakeholders at the second level generally are not the same as those at the first level. There may be, for example, additional customers for a sub-system or that are not customers for the system.

The lighter, generally upward directed lines are the feasibility, possibility, and query data that were generated as a result of the requirements being placed in the collections.

The dotted, upward directed lines represent the flow of requirement control data. This data is a record of the requirements in the hierarchy and their revision, source, and destination. This data is located in Template_Requirement Management Sheet.

6 APENDIX

A Requirements Working Group Information Report - Published in *Proceedings of the Seventh International Symposium of the INCOSE - Volume II, August 1997*; Prepared by the Requirements Working Group of the International Council on Systems Engineering, for information purposes only.

CMMI[SM] for Software Engineering (CMMI-SW, V1.1) - Staged Representation CMU/SEI-2002-TR-029 ESC-TR-2002-029 –August 2002

Twelve Requirements Basics for Project Success
Dr. Ralph R. Young, Northrop Grumman Information Technology Defense Group

Software Development Life Cycles: Outline for Developing a Traceability Matrix

By Diana Baldwin, AccuReg Inc.

Recommended Requirements Gathering Practices
Dr. Ralph R. Young, Northrop Grumman Information Technology

Approver	Reviewer	Creator
Nguyen Quang Hoa	QAs	Nguyen Thi Thu Ha