

## Chapter – Four

### Requirement Validation (RV)

**Requirement Validation is** a process of ensuring the specified requirements meet the customer needs. It's concerned with finding problems with the requirements.

#### Objectives of Validation

- Certifies that the requirements document is an acceptable description of the system to be implemented
- **Checks a requirements document for:**
  - Completeness and consistency
  - Conformance to standards
  - Requirements conflicts
  - Technical errors
  - Ambiguous requirements

#### Quality Assurance

The act of reviewing and auditing the project deliverables and activities to verify that they comply with the applicable standards, guidelines, and processes adopted by your organization is called quality assurance. Fundamentally, quality assurance attempts to answer the following questions: "Are you building the right thing?" and "Are you building it the right way?" Perhaps a more effective question to ask would be "Can we build this a better way?" because it would provide valuable feedback that developers could use to improve the way that they work.

A key concept in quality assurance is that quality is often in the eye of the beholder, indicating many aspects exist to software quality, including the following:

- Does it meet the needs of its users?
- Does it provide value to its stakeholders?
- Does it follow relevant standards?
- Is it easy to use by its intended users?

- Is it reasonably free of defects?
- Is it easy to maintain and to enhance?
- How easy will it integrate into the current technical environment?

## **Types of Validation Checks**

- During the requirements validation process, different types of checks should be carried out on the requirements.

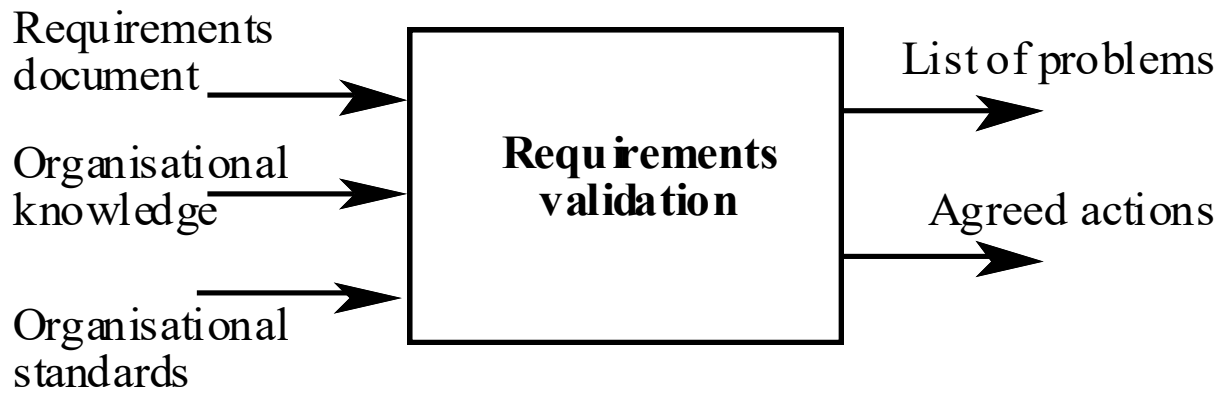
### **These checks include:**

- **Validity checks:** The functions proposed by stakeholders should be aligned with what the system needs to perform. You may find later that there are additional or different functions are required instead.
- **Consistency checks:** Requirements in the document shouldn't conflict or different description of the same function
- **Completeness checks:** The document should include all the requirements and constraints.
- **Realism checks:** Ensure the requirements can actually be implemented using the knowledge of existing technology, the budget, schedule, etc.
- **Verifiability:** Requirements should be written so that they can be tested. This means you should be able to write a set of tests that demonstrate that the system meets the specified requirements.

## **Analysis Validation**

- Analysis works with raw requirements as elicited from the system stakeholders
  - “Have we got the right requirements” is the key question to be answered at this stage
- Validation works with a final draft of the requirements document i.e. with negotiated and agreed requirements
  - “Have we got the requirements right” is the key question to be answered at this stage

## Validation Inputs and Outputs



### Validation inputs

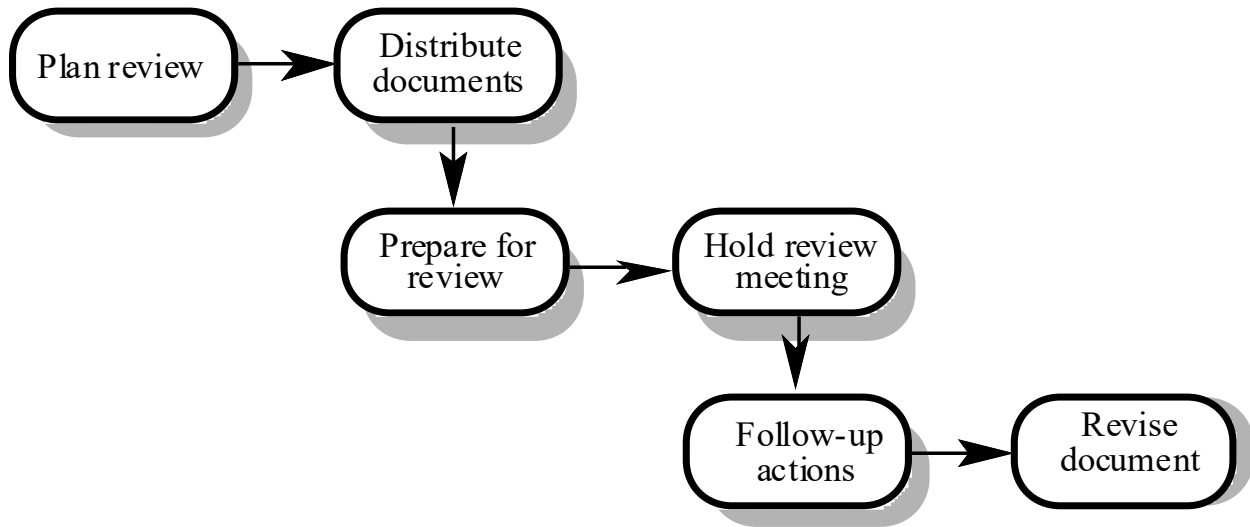
- **Requirements document**
  - Should be a complete version of the document, not an unfinished draft. Formatted and organised according to organisational standards
- **Organisational knowledge**
  - Knowledge, often implicit, of the organisation which may be used to judge the realism of the requirements
- **Organisational standards**
  - Local standards e.g. for the organisation of the requirements document

### Validation Outputs

- **Problem list**
  - List of discovered problems in the requirements document
- **Agreed actions**
  - List of agreed actions in response to requirements problems.
  - Some problems may have several corrective actions; some problems may have no associated actions

## Requirements Reviews

- A group of people read and analyse the requirements, look for problems, meet and discuss the problems and agree on actions to address these problems



**Figure: Requirements Review Process**

## Review Activities

- Plan review
  - The review team is selected and a time and place for the review meeting is chosen.
- Distribute documents
  - The requirements document is distributed to the review team members
- Prepare for review
  - Individual reviewers read the requirements to find conflicts, omissions, inconsistencies, deviations from standards and other problems.
- Hold review meeting
  - Individual comments and problems are discussed and a set of actions to address the problems is agreed.
- Follow-up actions
  - The chair of the review checks that the agreed actions have been carried out.
- Revise document

- The requirements document is revised to reflect the agreed actions. At this stage, it may be accepted or it may be re-reviewed

### **Problem Actions**

- **Requirements clarification**
  - The requirement may be badly expressed or may have accidentally omitted information which has been collected during requirements elicitation.
- **Missing information**
  - Some information is missing from the requirements document. It is the responsibility of the requirements engineers who are revising the document to discover this information from system stakeholders.
- **Requirements conflict**
  - There is a significant conflict between requirements. The stakeholders involved must negotiate to resolve the conflict.
- **Unrealistic requirement**
  - The requirement does not appear to be implementable with the technology available or given other constraints on the system. Stakeholders must be consulted to decide how to make the requirement more realistic.

### **Pre-Review Checking**

- Reviews are expensive because they involve a number of people spending time reading and checking the requirements document
- This expense can be reduced by using pre-review checking where one person checks the document and looks for straightforward problems such as missing requirements, lack of conformance to standards, typographical errors, etc.
- Document may be returned for correction or the list of problems distributed to other reviewers



- Is the document structured in a sensible way? Are the descriptions of requirements organised so that related requirements are grouped?
- Conformance to standards
  - Does the requirements document and individual requirements conform to defined standards? Are departures from the standards, justified?
- Traceability
  - Are requirements unambiguously identified, include links to related requirements and to the reasons why these requirements have been included?

### **Checklist Questions**

- Is each requirement uniquely identified?
- Are specialised terms defined in the glossary
- Does a requirement stand on its own or do you have to examine other requirements to understand what it means?
- Do individual requirements use the terms consistently
- Is the same service requested in different requirements? Are there any contradictions in these requests?
- If a requirement makes reference to some other facilities, are these described elsewhere in the document?
- Are related requirements grouped together? If not, do they refer to each other?

### **Requirements Problem Example**

“4. EDDIS will be configurable so that it will comply with the requirements of all UK and (where relevant) international copyright legislation. Minimally, this means that EDDIS must provide a form for the user to sign the Copyright Declaration statement. It also means that EDDIS must keep track of Copyright Declaration statements which have been signed/not-signed. Under no circumstances must an order be sent to the supplier if the copyright statement has not been signed.”

### **Problems**

- Incompleteness
  - What international copyright legislation is relevant?
  - What happens if the copyright declaration is not signed?
  - If a signature is a digital signature, how is it assigned?

- Ambiguity
  - What does signing an electronic form mean? Is this a physical signature or a digital signature?
- Standards
  - More than 1 requirement. Maintenance of copyright is one requirement; issue of documents is another

## **Prototyping**

The user interface (UI) of an application is the portion the user directly interacts with: screens, reports, documentation, and your software support staff. A user interface prototype is a user interface that has been "mocked up" using a computer language or prototyping tool, but it does not yet implement the full system functionality.

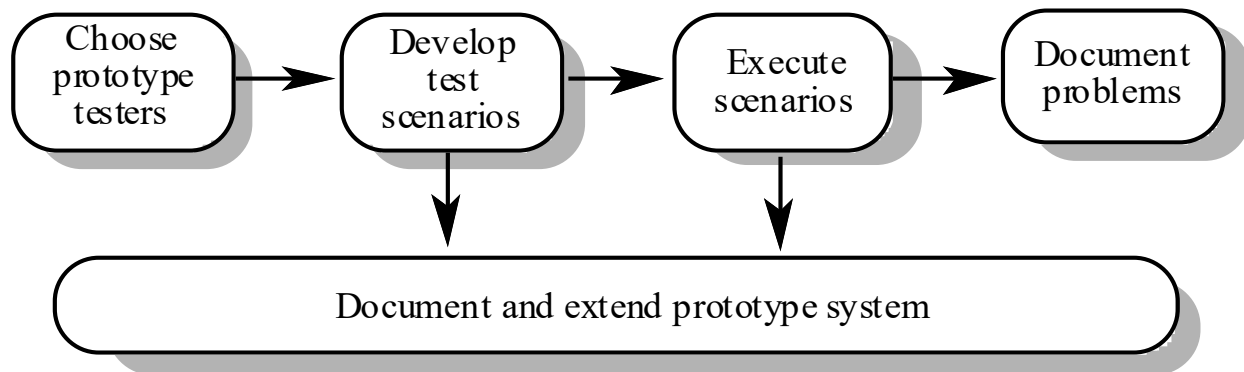
A prototype walkthrough is a testing process in which your users work through a series of usage scenarios to verify that a user prototype meets their needs. It is basically usage scenario testing applied to a user interface prototype instead of a domain model. The basic idea is that your users pretend the prototype is the real application and try to use it to solve real business problems described by the scenarios. Granted, they need to use their imaginations to fill in the functionality the application is missing (such as reading and writing objects from/to permanent storage), but, for the most part, this is a fairly straightforward process. Your users sit down at the computer and begin to work through the use cases. Your job is to sit there and observe them, looking for places where the system is difficult to use or is missing features. In many ways, prototype walkthroughs are a lot like user-acceptance tests, the only difference being you are working with the prototype instead of the real system.

- Prototypes for requirements validation demonstrate the requirements and help stakeholders discover problems
- Validation prototypes should be complete, reasonably efficient and robust. It should be possible to use them in the same way as the required system
- User documentation and training should be provided



## User-Interface Testing

UI testing is the verification that the UI follows the accepted standards chosen by your organization and the UI meets the requirements defined for it. User- interface testing is often referred to as graphical user interface (GUI) testing. UI testing can be something as simple as verifying that your application "does the right thing" when subjected to a defined set of user-interface events, such as keyboard input, or something as complex as a usability study where human-factors engineers verify that the software is intuitive and easy to use.



**Figure: Prototyping for Validation**

### Prototyping Activities

- **Choose prototype testers**
  - The best testers are **users** who are fairly experienced and who are open-minded about the use of new systems. End-users who do different jobs should be involved so that different areas of system functionality will be covered.
- **Develop test scenarios**
  - **Careful planning** is required to draw up a set of test scenarios which provide broad coverage of the requirements. End-users shouldn't just play around with the system as this may never exercise critical system features.
- **Execute scenarios**
  - The users of the **system work**, usually on their own, to try the system by executing the planned scenarios.
- **Document problems**
  - Its usually best to define some kind of **electronic or paper problem report** form which users fill in when they encounter a problem.

## User Manual Development

- Writing a user manual from the requirements forces a **detailed requirements analysis** and thus can reveal problems with the document
- **Information in the user manual**
  - Description of the functionality and how it is implemented
  - Which parts of the system have not been implemented
  - How to get out of trouble
  - How to install and get started with the system

## Model Validation/Reviews

A model review, also called a model walkthrough or a model inspection, is a validation technique in which your modeling efforts are examined critically by a group of your peers. The basic idea is that a group of qualified people, often both technical staff get together in a room to evaluate a model or document. The purpose of this evaluation is to determine whether the models not only fulfill the demands of the user community but also are of sufficient quality to be easy to develop, maintain, and enhance. When model reviews are performed properly, they can have a large payoff because they often identify defects early in the project, reducing the cost of fixing them

There are different "flavors" of model review. A requirements review is a type of model review in which a group of users and/or recognized experts review your requirements artifacts. The purpose of a user requirement review is to ensure your requirements accurately reflect the needs and priorities of your user community and to ensure your understanding is sufficient from which to develop software. Similarly an architecture review focuses on reviewing architectural models and a design review focuses on reviewing design models. As you would expect the reviewers are often technical staff.

*In general if you are going to hold a review, the following pointers should help you to make it effective:*

1. **Get the right people in the review.** You want people, and only those people, who know what they are looking at and can provide valuable feedback. Better yet, include them in your development efforts and avoid the review in the first place.

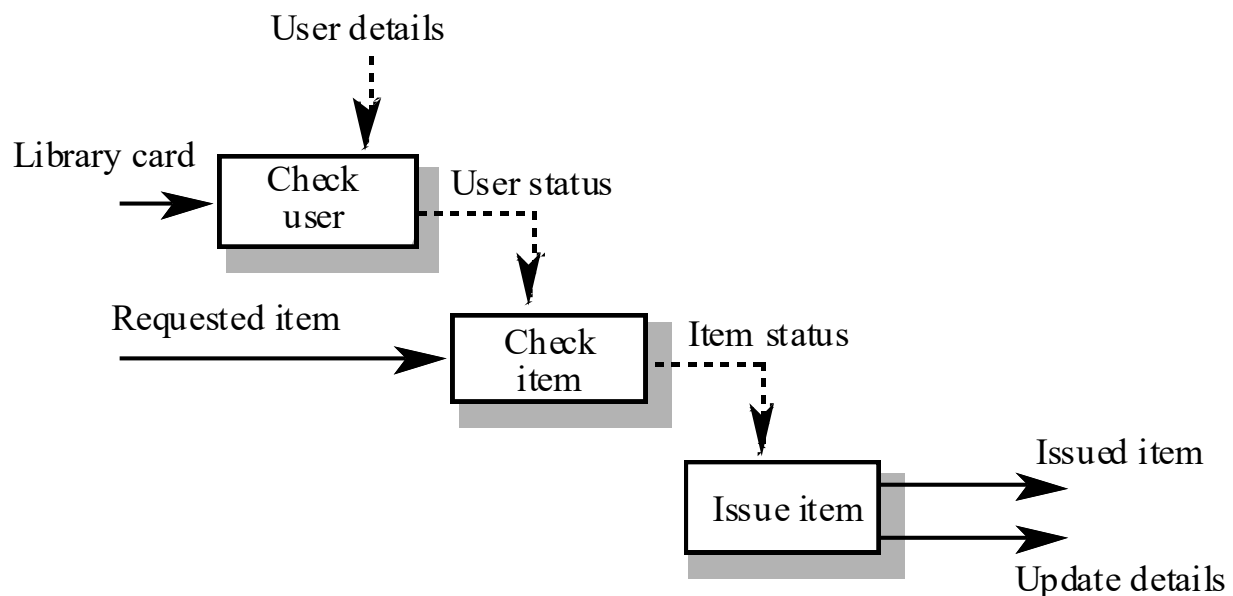
2. **Review working software, not models.** The traditional, near-serial development approach currently favored within many organizations provides little else for project stakeholders to look at during most of a project. However, because the iterative and incremental approach of agile development techniques tightens the development cycle you will find that user-acceptance testing can replace many model review efforts.
3. **Stay focused.** This is related to maximizing value: you want to keep reviews short and sweet. The purpose of the review should be clear to everyone; for example, if it is a requirements review do not start discussing database design issues. At the same time recognize that it is okay for an informal or impromptu model review to "devolve" into a modeling/working session as long as that effort remains focused on the issue at hand.
4. **Understand that quality comes from more than just reviews.** In application development, quality comes from developers who understand how to build software properly, who have learned from experience, and/or who have gained these skills from training and education. Reviews help you to identify quality deficits, but they will not help you build quality into your application from the outset. Reviews should be only a small portion of your overall testing and quality strategy.
5. **Set expectations ahead of time.** The expectations of the reviewers must be realistic if the review is to run smoothly. Issues that reviewers should be aware of are
  - The more detail a document has, the easier it is to find fault.
  - With an evolutionary approach your models are not complete until the software is ready to ship.
  - Agile developers are likely to be traveling light and therefore their documentation may not be "complete" either.
  - The more clearly defined a position on an issue, the easier it is to find fault.
  - Finding many faults may often imply a good, not a bad, job has been performed.
  - The goal is to find gaps in the work, so they can be addressed appropriately.
6. **Understand you cannot review everything.** You should prioritize your artifacts on a risk basis and review those that present the highest risk to your project if they contain serious defects.
7. **Focus on communication.** Reviews are vehicles for knowledge transfer, that they are opportunities for people to share and discuss ideas. However, working closely with your co-workers and project stakeholders while you are actually modeling is even more effective

for this purpose than reviews. This philosophy motivates agile developers to avoid formal reviews, due to their restrictions on how people are allowed to interact, in favor of other model validation techniques.

8. **Put observers to work.** People will often ask to observe a review either to become trained in the review process or to get updated on the project.

- **Objectives of model validation**

- To demonstrate that **each model** is **self-consistent**
- If there are several models of the system, to demonstrate that these are **internally and externally consistent**
- To demonstrate that the models accurately **reflect** the **real requirements** of system stakeholders
- Some checking is possible with automated tools
- Paraphrasing the model is an effective checking technique



**Figure: Data-flow diagram for Issue**

**Paraphrased Description**

<b>Check user</b>	
Inputs and sources	User's library card from end-user
Transformation function	Checks that the user is a valid library user
Transformation outputs	The user's status
Control information	User details from the database
<b>Check item</b>	
Inputs and sources	The user's status from Check user
Transformation function	Checks if an item is available for issue
Transformation outputs	The item's status
Control information	The availability of the item
<b>Issue item</b>	
Inputs and sources	<i>None</i>
Transformation function	Issues an item to the library user. Items are stamped with a return date.
Transformation outputs	The item issued to the end user Database update details
Control information	Item status - items only issued if available

## Requirements Testing

- Each requirement should be testable i.e. it should be possible to define tests to check whether or not that requirement has been met.
- Inventing requirements tests is an effective validation technique as missing or ambiguous information in the requirements description may make it difficult to formulate tests
- Each functional requirement should have an associated test

## Test cases

- What usage scenario might be used to check the requirement?
- Does the requirement, on its own, include enough information to allow a test to be defined?
- Is it possible to test the requirement using a single test or are multiple test cases required?
- Could the requirement be re-stated to make the test cases more obvious?

## Test record form

- **The requirement's identifier**
  - There should be at least one for each requirement.
- **Related requirements**
  - These should be referenced as the test may also be relevant to these requirements.

- **Test description**
  - A brief description of the test and why this is an objective requirements test. This should include system inputs and corresponding outputs.
- **Requirements problems**
  - A description of problems which made test definition difficult or impossible.
- **Comments and recommendations**
  - These are advice on how to solve requirements problems which have been discovered.

### **Requirements test form**

**Requirements tested:** 10.(iv)

**Related requirements:** 10.(i), 10.(ii), 10.(iii), 10.(vi), 10.(vii)

**Test applied:** For each class of user, prepare a login script and identify the services expected for that class of user.

The results of the login should be a web page with a menu of available services.

**Requirements problems:** We don't know the different classes of EDDIS user and the services which are available to each user class. Apart from the administrator, are all other EDDIS users in the same class?

**Recommendations:** Explicitly list all user classes and the services which they can access.

## Hard-to-test Requirements

- **System requirements**
  - Requirements which apply to the system as a whole. In general, these are the **most difficult requirements** to validate irrespective of the method used as they may be influenced by any of the functional requirements. Tests, which are not executed, cannot test for non-functional system-wide characteristics such as usability.
- **Exclusive requirements**
  - These are requirements which exclude specific behaviour. For example, a requirement may state that system failures must never corrupt the system database. It is not possible to test such a requirement exhaustively.
- **Some non-functional requirements**
  - Some non-functional requirements, such as reliability requirements, can only be tested with a large test set. Designing this test set does not help with requirements validation.

### Key points

- Requirements validation should focus on checking the final draft of the requirements document for conflicts, omissions and deviations from standards.
- Inputs to the validation process are the requirements document, organisational standards and implicit organisational knowledge. The outputs are a list of requirements problems and agreed actions to address these problems.
- Reviews involve a group of people making a detailed analysis of the requirements.
- Review costs can be reduced by checking the requirements before the review for deviations from organisational standards. These may result from more serious requirements problems.
- Checklists of what to look for may be used to drive a requirements review process.
- Prototyping is effective for requirements validation if a prototype has been developed during the requirements elicitation stage.
- Systems models may be validated by paraphrasing them. This means that they are systematically translated into a natural language description.
- Designing tests for requirements can reveal problems with the requirements. If the requirement is unclear, it may be impossible to define a test for it.