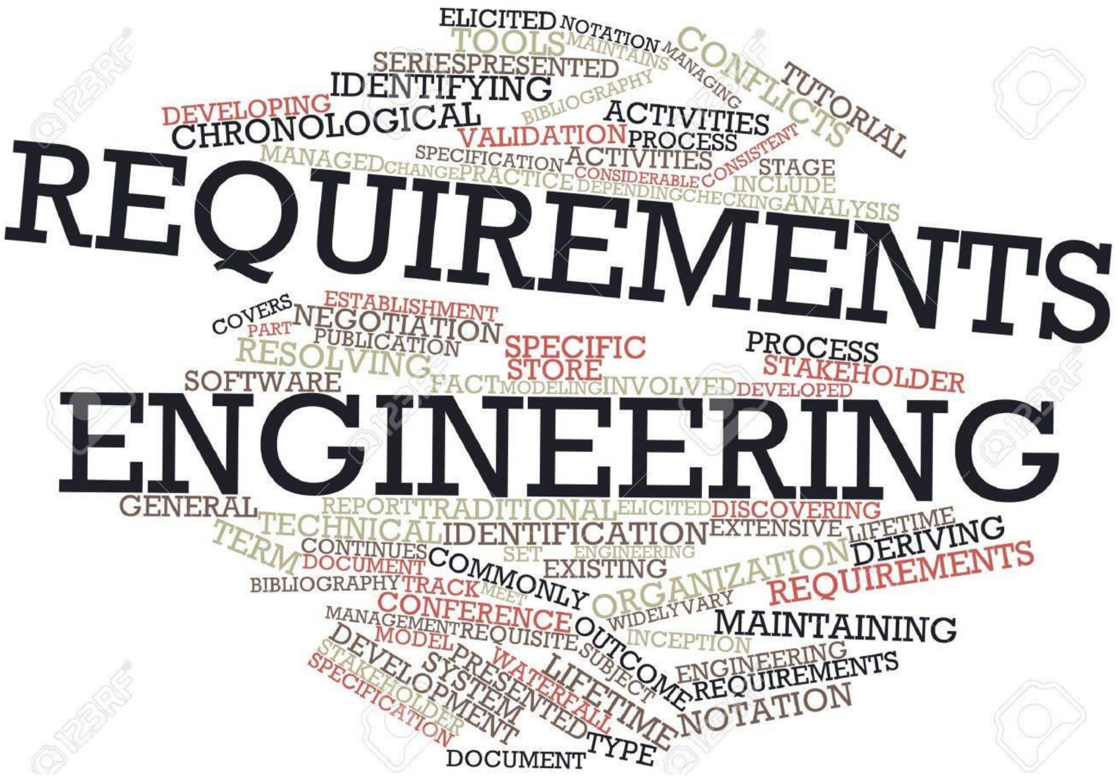


SWE 215: Software Requirements Engineering

Lecture 11

Requirements Validation

Course Topics

- ~~Why Requirements Engineering?~~
- ~~Introduction to Requirements~~
- ~~RE in Software Development Life Cycles~~
- ~~System Vision, Context, and RE Framework~~
- ~~Fundamentals of Goal Orientation~~
- ~~Fundamentals of Scenarios~~
- ~~Requirements Discovery~~
- ~~User Stories and Agile Estimation~~
- ~~Features Prioritization~~
- ~~Requirements Negotiation~~
- Requirements Validation
- Fundamentals of Requirements Management

Lecture Objectives

- Requirements Risk Management
- Validation vs. Verification
- Requirements V&V Techniques:
- Requirements Reviews
- Prototyping

Requirements Risks

Requirements can be **inadequate** in many ways including:

- Inaccurate or incomplete **stakeholder identification**
- Insufficient requirements **validation and verification**
- **Incomplete, inconsistent or incorrect requirements**
- **Incorrectly ranked requirements**

Requirements risk management involves the **proactive analysis**, **identification**, **monitoring**, and **mitigation** of any factors that can threaten the integrity of the requirements engineering process.

Example of issues in Requirements

A set of requirements for an electric water heater controller:

- If 70 ° <temperature <100 °, then the system shall output 3000 watts.
- If 100 ° <temperature <130 °, then the system shall output 2000 watts.
- If 120 ° <temperature <150 °, then the system shall output 1000 watts.
- If 150° <temperature, then the system shall output 0 watts.

Example of issues in Requirements

A set of requirements for an electric water heater controller:

- If 70 ° <temperature <100 °, then the system shall output 3000 watts.
- If 100 ° <temperature <130 °, then the system shall output 2000 watts.
- If 120 ° <temperature <150 °, then the system shall output 1000 watts.
- If 150° <temperature, then the system shall output 0 watts.

Some identified Issues:

- The set of requirements is **incomplete** because the behavior for temperature <0° is not defined.
- The requirements are also **inconsistent—for** example, what happens when temperature = 125 °?
- The requirements are also **unclear** because the temperatures given are not specified as being in degree Fahrenheit or degree Celsius.

Requirements Verification and Validation (V & V)

- Requirements validation and verification involves **review**, **analysis**, and **testing** to ensure that a system complies with its requirements.

- Compliance pertains to both **functional** and **nonfunctional** requirements.

- **Validation**: "Are we building the right product?"
- **Verification:** "Are we building the product right?"

- In other words, **validation** involves **fully understanding of customer intent** and **verification** involves **satisfying the customer intent**.

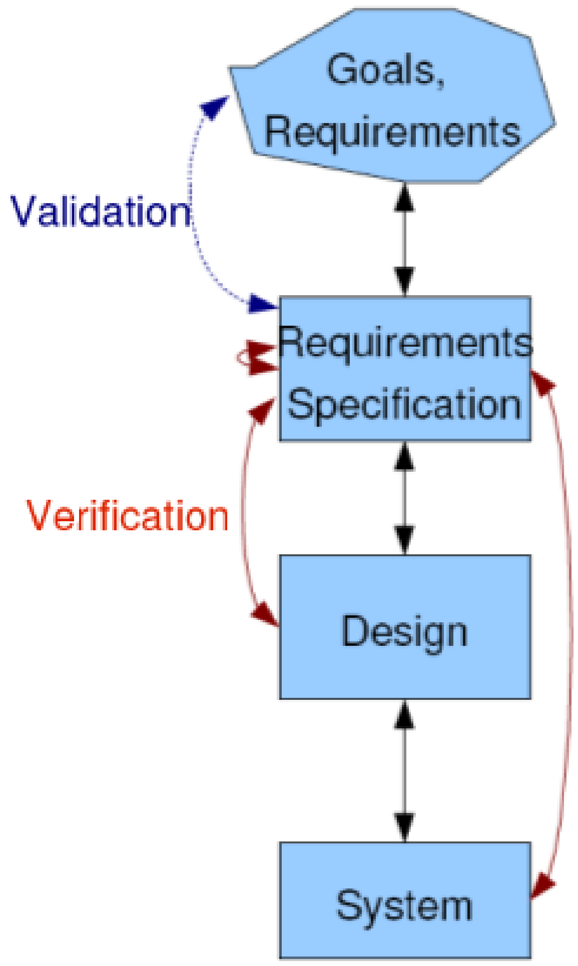
Verification and Validation (V & V)

Validation

Ensures that the software being developed (or changed) will **satisfy its stakeholders**
Checks the software requirements specification against stakeholders goals and requirements

Verification

Checks consistency of the software requirements specification artefacts and other software development products (design, implementation, ...) against the specification



Requirements Validation and Verification Objectives

- 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

Analysis and Validation

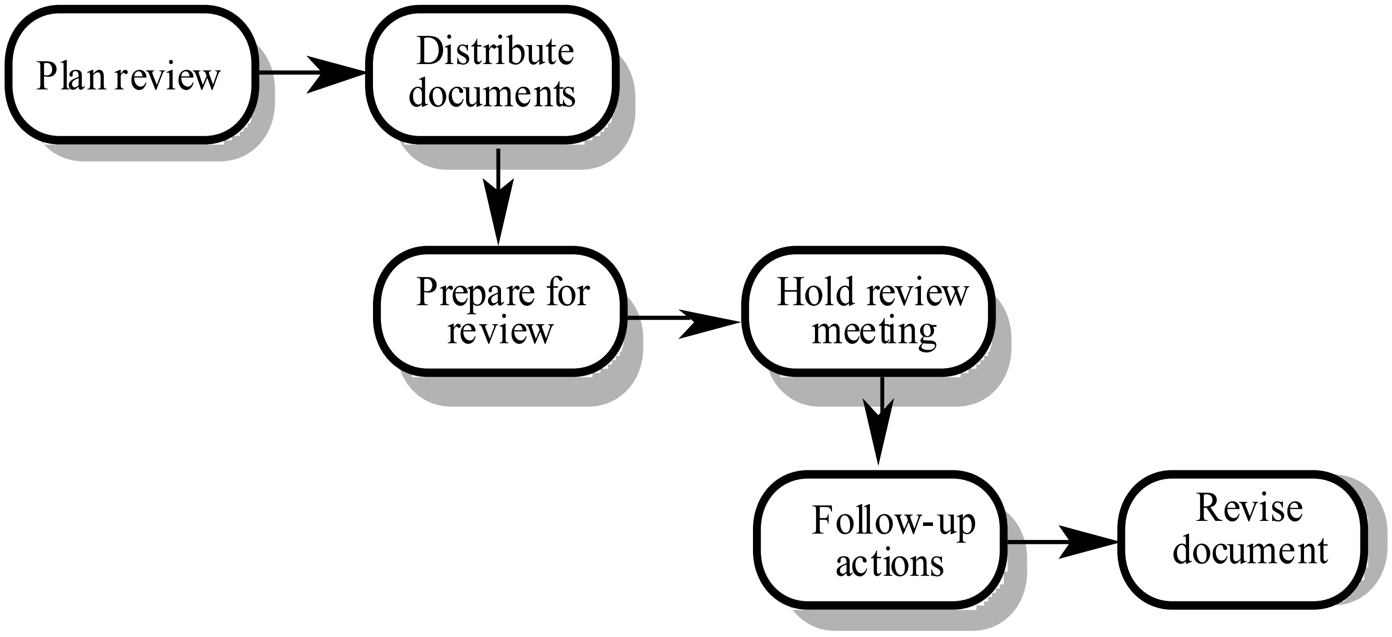
- **Analysis** works with **raw requirements** as elicited from the system stakeholders.
- "**Have we got the right requirements?**" is the key question to be answered at this stage
- **Validation** works with a final draft of the requirements document i.e., with negotiated and agreed requirements
-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

Requirements V&V Techniques

1. **Requirements Reviews/Inspections**
2. **Prototyping**

Requirements Reviews

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

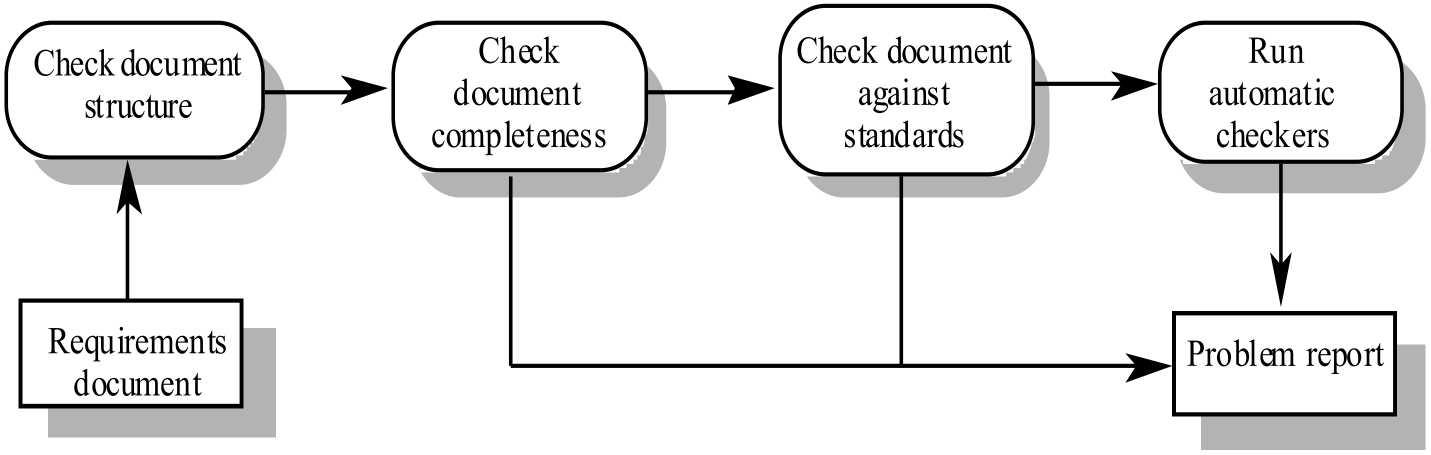


Review Activities

- **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 upon.
- **Follow-up actions:** The chair of the review checks that the agreed upon actions have been carried out.
- **Revise document:** The requirements document is revised to reflect the agreed upon actions. At this stage, it may be accepted or it may be re-reviewed

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



Review team membership

- **Reviews** should involve a **number of stakeholders** drawn from **different backgrounds**
- People from different backgrounds **bring different skills** and **knowledge to the review**
- Stakeholders **feel involved in the RE process** and develop an understanding of the needs of other stakeholders
- Review team should always involve **at least a domain expert and an end-user**

Review/Inspection checklists

Understandability: Can readers of the document understand what the requirements mean?

Redundancy: Is information unnecessarily repeated in the requirements document?

Completeness: Does the checker know of any missing requirements or is there any information missing from individual requirement descriptions?

Ambiguity: Are the requirements expressed using terms which are clearly defined? Could readers from different backgrounds make different interpretations of the requirements?

Review/Inspection checklists

Consistency: Do the descriptions of different requirements include contradictions? Are there contradictions between individual requirements and overall system requirements?

Conformance to standards: Does the requirements document and individual requirements conform to defined standards? Are departures from the standards, justified?

Organization: Is the document structured in a sensible way? Are the descriptions of requirements organized so that related requirements are grouped?

Traceability: Are requirements unambiguously identified, include links to related requirements and to the reasons why these requirements have been included?

Example of a checklist for inspecting Use Case models

1. Actors

1.1. Are there any actors that are not defined in the use case model, that is, will the system communicate with any other systems, hardware or human users that have not been described?
1.2. Are there any superfluous actors in the use case model, that is, human users or other systems that will not provide input to or receive output from the system?
1.3. Are all the actors clearly described, and do you agree with the descriptions?
1.4. Is it clear which actors are involved in which use cases, and can this be clearly seen from the use case diagram and textual descriptions? Are all the actors connected to the right use cases?

2. The use cases

2.1. Is there any missing functionality, that is, do the actors have goals that must be fulfilled, but that have not been described in use cases?
2.2. Are there any superfluous use cases, that is, use cases that are outside the boundary of the system, do not lead to the fulfillment of a goal for an actor or duplicate functionality described in other use cases?
2.3. Do all the use cases lead to the fulfillment of exactly one goal for an actor, and is it clear from the use case name what is the goal?
2.4. Are the descriptions of how the actor interacts with the system in the use cases consistent with the description of the actor?
2.5. Is it clear from the descriptions of the use cases how the goals are reached and do you agree with the descriptions?

Example of a Checklist for inspecting Use Case models

3. The description of each use case

3.1. Is expected input and output correctly defined in each use case; is the output from the system defined for every input from the actor, both for normal flow of events and variations?

3.2. Does each event in the normal flow of events relate to the goal of its use case?

3.3. Is the flow of events described with concrete terms and measurable concepts and is it described at a suitable level of detail without details that restrict the user interface or the design of the system?

3.4. Are there any variants to the normal flow of events that have not been identified in the use cases, that is, are there any missing variations?

3.5. Are the triggers, starting conditions, for each use case described at the correct level of detail?

3.6. Are the pre- and post-conditions correctly described for all use cases, that is, are they described with the correct level of detail, do the pre- and post conditions match for each of the use cases and are they testable?

4. Relation between the use cases:

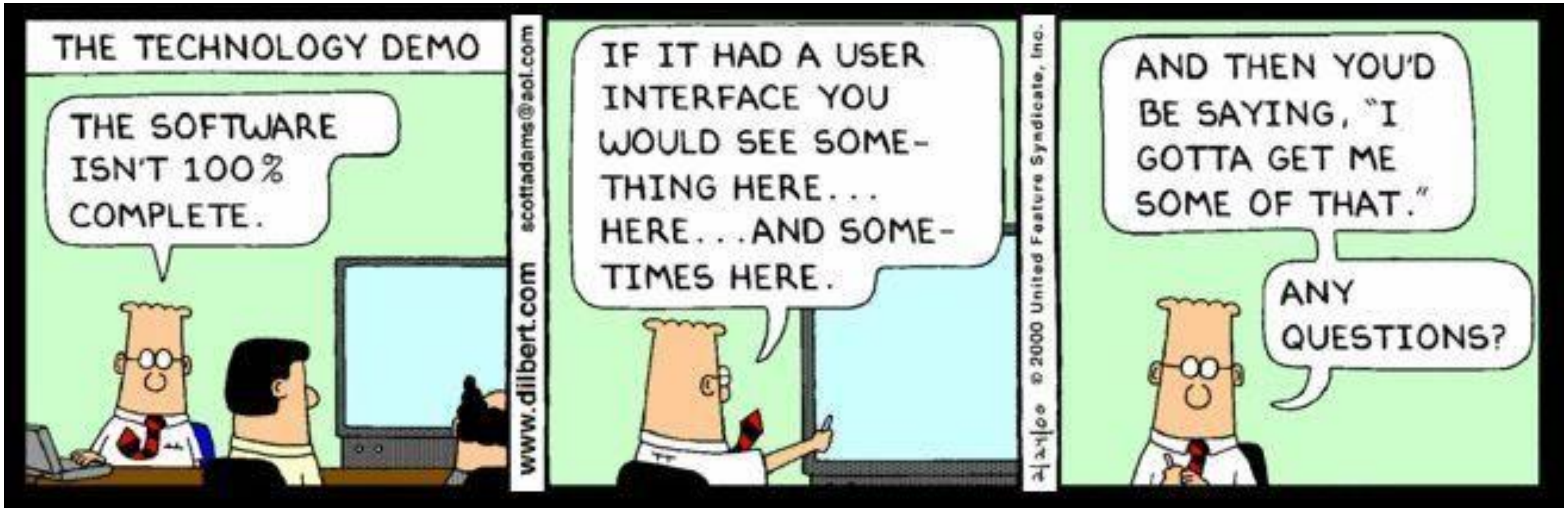
4.1. Do the use case diagram and the textual descriptions match?

4.2. Has the include-relation been used to factor out common behavior?

4.3. Does the behavior of a use case conflict with the behavior of other use cases?

4.4. Are all the use cases described at the same level of detail?

Prototyping


THE TECHNOLOGY DEMO
THE SOFTWARE ISN'T 100% COMPLETE.
IF IT HAD A USER INTERFACE YOU WOULD SEE SOMETHING HERE... HERE... AND SOMETIMES HERE.
AND THEN YOU'D BE SAYING, "I GOTTA GET ME SOME OF THAT."
ANY QUESTIONS?
www.dilbert.com scottadams@aol.com
© 2000 United Feature Syndicate, Inc.

➢ Dilema:

You can't evaluate design until it's built

– But…

- After building, changes to the design are difficult
- **Simulate the design, in low-cost manner**

Prototyping Dimensions

1. Representation
– Can be just textual description or can be visuals and diagrams

2. Scope
– Is is just the interface (mock-up) or does it include some computational component?

3. Executability
– Can the prototype be “run”?

4. Maturation
– What are the stages of the product as it comes along?
(Throw-away vs. Evolutionary)

Types of Prototypes

- Prototypes have different shapes and sizes:
  - Low Fidelity vs. High Fidelity
  - Horizontal vs. vertical
  - Evolutionary vs. throwaway

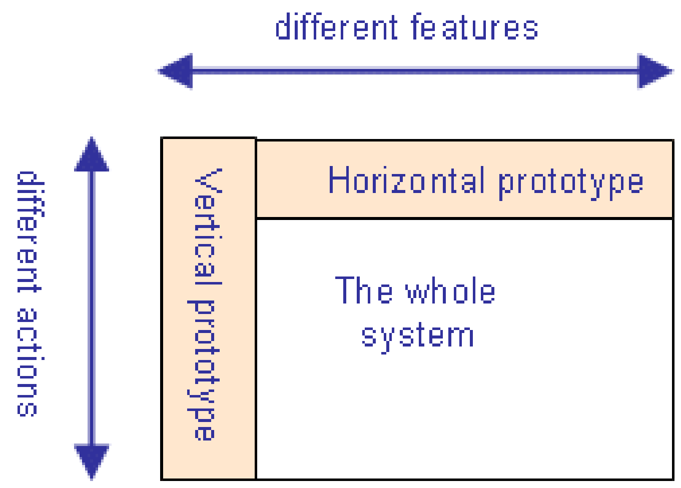
Low Fidelity vs. High Fidelity

- **Low-fidelity prototype**
  - Far from final form of system, such as paper, drawings, etc.
  - Sketchy and incomplete, that has some characteristics of the target product but is otherwise simple

- **High-fidelity prototype**
  - Close to final form of system, much more realistic to actual application.
  - A computer-based interactive representation of the product.

Horizontal vs. Vertical Prototyping (1)

Horizontal Prototype: Provides a broad view of an entire system or subsystem, focusing on user interaction. **Example**: all first and second level menu commands.



Vertical Prototype: A more complete elaboration of few functions.

Horizontal vs. Vertical Prototyping (2)

Horizontal Prototype: Useful for:

- Confirmation of user interface requirements and system scope,
- Develop preliminary estimates of development time, cost and effort.

Vertical Prototype: Useful for **obtaining detailed requirements** for **a given function**, with the following benefits:

- Refinement database design,
- Clarify complex requirements by drilling down to actual system functionality.

Throwaway vs. Evolutionary Prototyping

Throwaway or **Rapid Prototyping:**

- Creation of a model that will eventually be discarded rather than becoming part of the final delivered software.
- It can be done quickly → quick feedback
- Making changes early in the development lifecycle is **extremely cost effective**

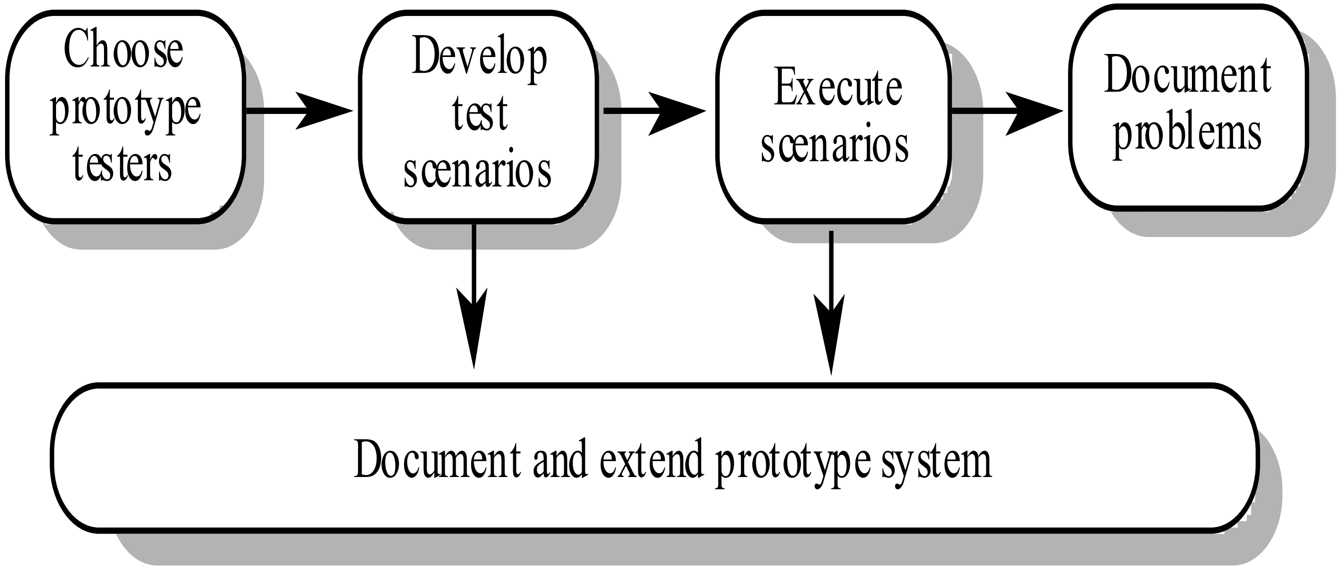
Evolutionary Prototyping (also known as **breadboard** prototyping):

- Build a very robust prototype in a structured manner and constantly refine it.
- Developers can focus on developing parts of the system that they understand instead of working on developing a whole system.

Prototyping for Requirements Validation

- 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.**

Prototyping for Validation



Prototyping Validation Steps

- **Choose prototype testers**
  - The best testers are **users** who are fairly **experienced** and who are **open-minded** about the use of new systems.
- **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 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 functionalities
  - How to get out of trouble
  - How to install and get started with the system

Models V&V

- Validation of system models is an essential part of the validation process

- Objectives of model V&V:
  - 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**

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 Case Definition

- What **usage scenarios** 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 requirement 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 advices on how to solve requirements problems which have been discovered.

Key points

- Requirements validation should **focus on checking the final draft** of the requirements document for **conflicts**, **omissions** and **deviations from standards.**
- **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 organizational standards.
- **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**.
- 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.