MODULE- 2

REQUIREMENTS ENGINEERING

- **Requirements engineering** provides the appropriate mechanism for understanding what the customer wants, analysing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.
- It encompasses seven distinct tasks: **inception, elicitation, elaboration, negotiation, specification, validation, and management.**
- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.

Inception: At project inception, we establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

Elicitation: Ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis.

Christel and Kang identify a number of problems that are encountered as elicitation occurs.

- Problems of scope: The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
- Problems of understanding: The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be "obvious," specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or untestable.
- Problems of volatility: The requirements change over time.

To help overcome these problems, you must approach requirements gathering in an organized manner.

Elaboration:

- ✓ The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- ✓ This task focuses on developing a refined requirements model that identifies various aspects of software function, behaviour, and information.
- ✓ Elaboration is driven by the creation and refinement of user scenarios that describe how the end user (and other actors) will interact with the system.
- ✓ Each user scenario is parsed to extract analysis classes—business domain entities that are visible to the end user. The attributes of each analysis class are defined, and the services that are required by each class are identified.
- ✓ The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.

Negotiation:

- ✓ It isn't unusual for customers and users to ask for more than can be achieved, given limited business resources.
- ✓ It's also relatively common for different customers or users to propose conflicting requirements, arguing that their version is "essential for our special needs."
- ✓ We have to reconcile these conflicts through a process of negotiation.
- ✓ Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
- ✓ Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

Specification:

- ✓ A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
- ✓ Some suggest that a "standard template" should be developed and used for a specification, arguing that this leads to requirements that are presented in a consistent and therefore more understandable manner.

INFO

Software Requirements Specification Template

A *software requirements specification* (SRS) is a document that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at **www.processimpact.com/process_assets/srs_template.doc**) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

Table of Contents
Revision History

1. Introduction
1.1 Purpose
1.2 Document Conventions
1.3 Intended Audience and Reading Suggestions
1.4 Project Scope
1.5 References

2. Overall Description
2.1 Product Perspective
2.2 Product Features
2.3 User Classes and Characteristics
2.4 Operating Environment
2.5 Design and Implementation Constraints
2.6 User Documentation
2.7 Assumptions and Dependencies

3. System Features
3.1 System Feature 1
3.2 System Feature 2 (and so on)

4. External Interface Requirements
4.1 User Interfaces
4.2 Hardware Interfaces
4.3 Software Interfaces
4.4 Communications Interfaces

5. Other Nonfunctional Requirements
5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted earlier in this sidebar.

Validation:

- ✓ The work products produced as a consequence of requirements engineering are assessed for quality during a validation step.
- ✓ Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.
- ✓ The primary requirements validation mechanism is the technical review.
- ✓ The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies (a major problem when large products or systems are engineered), conflicting requirements, or unrealistic (unachievable) requirements.

INFO

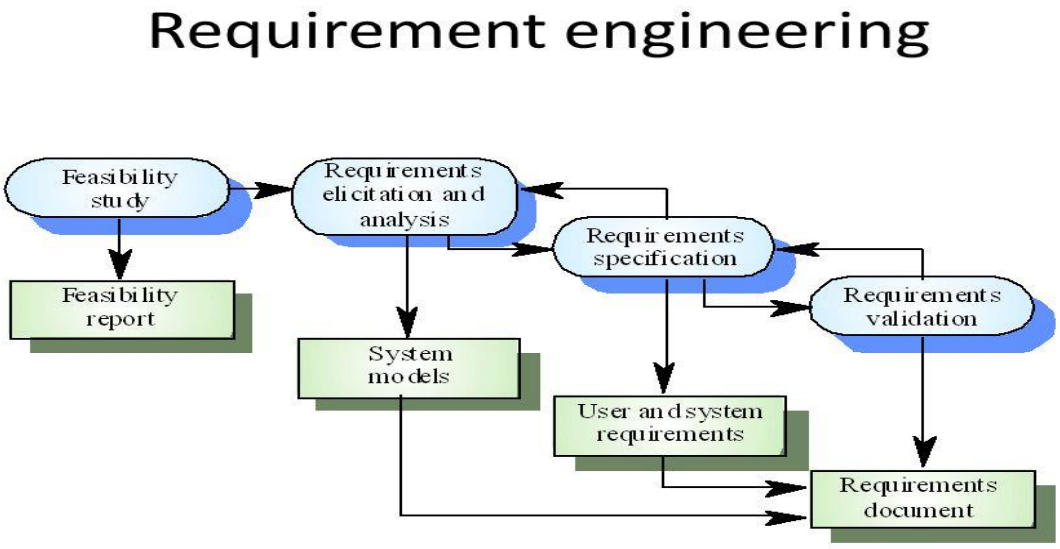
Requirements Validation Checklist

It is often useful to examine each requirement against a set of checklist questions. Here is a small subset of those that might be asked:

- Are requirements stated clearly? Can they be misinterpreted?
- Is the source (e.g., a person, a regulation, a document) of the requirement identified? Has the final statement of the requirement been examined by or against the original source?
- Is the requirement bounded in quantitative terms?
- What other requirements relate to this requirement? Are they clearly noted via a cross-reference matrix or other mechanism?
- Does the requirement violate any system domain constraints?
- Is the requirement testable? If so, can we specify tests (sometimes called validation criteria) to exercise the requirement?
- Is the requirement traceable to any system model that has been created?
- Is the requirement traceable to overall system/product objectives?
- Is the specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?
- Has an index for the specification been created?
- Have requirements associated with performance, behavior, and operational characteristics been clearly stated? What requirements appear to be implicit?

Requirements management:

- ✓ Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system.
- ✓ Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.
- ✓ Many of these activities are identical to the software configuration management (SCM) techniques.



INITIATING THE REQUIREMENTS ENGINEERING PROCESS

Steps to be followed:

1. Identifying Stakeholders
2. Recognizing Multiple Viewpoints
3. Working toward Collaboration
4. Asking the First Questions

Identifying Stakeholders:

- A stakeholder is "anyone who benefits in a direct or indirect way from the system which is being developed"
- They are business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, support and maintenance engineers, and others.
- Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.
- At inception, you should create a list of people who will contribute input as requirements are elicited.

Recognizing Multiple Viewpoints

- In the process of developing a new system, various stakeholder groups, including marketing, business managers, end users, software engineers, and support engineers, each have unique requirements based on their specific interests and needs.
- These requirements can range from marketable features and budget compliance to user-friendly interfaces, technical infrastructure, and maintainability.
- As these diverse and sometimes conflicting requirements are collected during the requirements engineering process, they need to be effectively categorized and managed.
- This categorization helps in identifying inconsistencies and conflicts, allowing decision-makers to select a consistent and unified set of requirements for the system's development.

Working toward Collaboration

- In a software project involving multiple stakeholders, differing opinions on requirements are common. Effective collaboration among stakeholders and software engineers is crucial for a successful system.
- The role of a requirements engineer is to pinpoint areas where stakeholders agree (commonality) and areas of disagreement or inconsistency.
- Handling conflicting requirements, where one stakeholder's needs clash with another's, is particularly challenging.
- Collaboration does not necessarily involve making decisions by committee; rather, stakeholders may provide input on requirements, but a strong project champion (like a business manager or senior technologist) often makes the final decisions on which requirements are included in the project.

Asking the First Questions

- The first set of context-free questions focuses on the customer and other stakeholders, the overall project goals and benefits.
- For example, you might ask:
  Who is behind the request for this work?
  Who will use the solution?
  What will be the economic benefit of a successful solution?
  Is there another source for the solution that you need?
- These questions help to identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development.
- The next set of questions enables you to gain a better understanding of the problem and allows the customer to voice his or her perceptions about a solution:
  How would you characterize "good" output that would be generated by a successful solution?
  What problem(s) will this solution address?
  Can you show me (or describe) the business environment in which the solution will be used?
  Will special performance issues or constraints affect the way the solution is approached?

- The final set of questions focuses on the effectiveness of the communication activity itself:
  Are you the right person to answer these questions?
  Are your answers "official"?
  Are my questions relevant to the problem that you have?
  Am I asking too many questions?
  Can anyone else provide additional information?
  Should I be asking you anything else?

ELICITING REQUIREMENTS

- Requirements elicitation (also called requirements gathering) combines elements of problem solving, elaboration, negotiation, and specification.
- In order to encourage a collaborative, team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements

Collaborative Requirements Gathering

Different approaches to collaborative requirements gathering have been proposed. And the following are basic guidelines:

- Meetings are conducted and attended by both software engineers and other stakeholders.
- Rules for preparation and participation are established.
- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

SAFEHOME

Conducting a Requirements Gathering Meeting

The scene: A meeting room. The first requirements gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator (pointing at whiteboard): So that's the current list of objects and services for the home security function.

Marketing person: That about covers it from our point of view.

Vinod: Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

Marketing person: Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints?

Jamie: It does, both technical and legal.

Production rep: Meaning?

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

Doug: Very true.

Marketing: But we still need that . . . just be sure to stop an outsider from getting in.

Ed: That's easier said than done and . . .

Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here.

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

Quality Function Deployment

- Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
- QFD "concentrates on maximizing customer satisfaction from the software engineering process"
- To accomplish this, QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.
- QFD identifies three types of requirements:

  Normal requirements: The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied. Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance.

  Expected requirements: These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction.

  Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation.

Exciting requirements: These features go beyond the customer's expectations and prove to be very satisfying when present. For example, software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multitouch screen, visual voice mail) that delight every user of the product.

Jamie: What if I forget my password?

Facilitator (interrupting): Good point, Jamie, but let's not address that now. We'll make a note of that and call it an *exception*. I'm sure there'll be others.

Marketing person: After I enter the passwords, a screen representing all *SafeHome* functions will appear. I'd select the home security function. The system might request that I verify who I am, say, by asking for my address or phone number or something. It would then display a picture of the security system control panel along with a list of functions that I can perform—arm the system, disarm the system, disarm one or more sensors. I suppose it might also allow me to reconfigure security zones and other things like that, but I'm not sure.

(As the marketing person continues talking, Doug takes copious notes; these form the basis for the first informal usage scenario. Alternatively, the marketing person could have been asked to write the scenario, but this would be done outside the meeting.)

Usage Scenarios

Gathering system requirements helps create a vision of its functions, but diving into technical work requires understanding how users will use these functions. To do this, developers and users make Usage scenarios showing how the system will be used.

SAFEHOME

Developing a Preliminary User Scenario

The scene: A meeting room, continuing the first requirements gathering meeting.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator: We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

Jamie: How?

Facilitator: We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

Marketing person: Um . . . well, this is the kind of thing I'd do if I was away from home and I had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

Facilitator (smiling): That's the reason you'd do it . . . tell me how you'd actually do this.

Marketing person: Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user id and . . .

Vinod (interrupting): The Web page would have to be secure, encrypted, to guarantee that we're safe and . . .

Facilitator (interrupting): That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK?

Vinod: No problem.

Marketing person: So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords.

Elicitation Work Products

- The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built.
- For most systems, the work products include
  - A statement of need and feasibility.
  - A bounded statement of scope for the system or product.
  - A list of customers, users, and other stakeholders who participated in requirements elicitation.
  - A description of the system's technical environment.
  - A list of requirements (preferably organized by function) and the domain constraints that apply to each.
  - A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
  - Any prototypes developed to better define requirements