Module - 1

- Software requirements specification
- Eliciting Software requirements
- Requirement specifications
- Software requirements engineering co
- Requirements modelling
- Use cases and User stories.
- Requirements documentation.

Software Requirements Specificatio (SRS)

- A **requirement** can range from a high-level, abstract statement of a constraint to a detailed, formal specification.
- SRS is the set of activities designed to capture behavioral and non-b aspects of the system in the SRS document.
- The goal of the SRS activity, and the resultant documentation, is to complete description of the system’s behavior without describing th structure.
- This aspect is easily stated, but difficult to achieve.

- Software specifications provide the basis for analyzing the requir validating that they are the stakeholder’s intentions, defining wha designers have to build, and verifying that they have done so cor
- SRSs allow us to know the motivation for development of the so system.
- Software requirements also help software engineers manage the e of the software over time and across families of related software products.

Requirements Engineering Activiti

- First, we must **elicit the requirements**, which is a form of discovery.
- The term “gathering” is also used to describe the process of collecting requirements.
- Often, requirements are deeply hidden, expressed incorrectly by stakeh contradictory, and complex.
- Eliciting requirements is one of the hardest jobs for the requirements e

- **Modeling requirements** involves representing the requirements in form.
- Words, pictures, and mathematical formulas can all be used but it is easy to effectively model requirements.
- **Analyzing requirements** involves determining if the requirements correct or have certain other properties such as consistency, complet sufficient detail, and so on.
- Finally, requirements change all the time and the requirements engi must be equipped to deal with this eventuality.

- SRSs are usually classified in terms of their level of abstractio

1. user requirements

2. system requirements

3. software design specifications

- **User requirements specifications** are usually used to attract bidders in the form of a request for proposal (RFP).
  - User requirements may contain abstract statements in natural lang with accompanying informal diagrams.
  - User requirements specify functional and non-functional requiren they pertain to externally visible behavior in a form understandab clients and system users.

- **System level requirements**, or SRSs mentioned previously, are det
  descriptions of the services and constraints.
  - Systems requirements are derived from analysis of the user requirements.
  - Systems requirements should be structured and precise because
    as a contract between client and contractor (and can literally be
    in court).

- **Software design specifications** are usually the most detailed lev requirements specifications that are written and used by softwar engineers as a basis for the system’s architecture and design.

- **The user needs are usually called “functional requirements” external constraints are called “non-functional requirements**

- **Functional requirements** describe the services the system should pr
- Sometimes the functional requirements state what the system should
- Functional requirements can be high-level and general or detailed, expressing inputs, outputs, exceptions, and so on.
- **Non-functional requirements** are imposed by the environment in w system is to exist.
- These requirements could include timing constraints, quality propert standard adherence, programming languages to be used, compliance laws, and so on.

Functional requirements

- ***Interface specifications***
- ***Performance requirements***
- ***Logical database requirements***
- ***System attribute requirements***

Non-Functional requirer

- ***Domain requirements***
- ***Design constraint req***

- **Interface specifications** are functional software requirements specifi
  terms of interfaces to operating units.

- Most systems must operate with other systems and the operating inte
  must be specified as part of the requirements.

- There are three types of interface that may have to be defined: proced
  interfaces, data structures that are exchanged , and data representatio

- **System attribute requirements** are functional requirements that incl
  reliability, availability, security, maintainability, and portability.

- **Performance requirements** are functional requirements that ir
  the static and dynamic numerical requirements placed on the so
  or on human interaction with the software as a whole.

- **Logical database requirements** are functional requirements th
  include the types of information used by various functions such
  frequency of use, accessing capabilities, data entities and their
  relationships, integrity constraints, and data retention requireme

- **Domain requirements** are a type of non-functional requiremen
  which the application domain dictates or derives.
  - For example, in the baggage inspection system, industry star
    and restrictions on baggage size and shape will place certain
    constraints on the system.
- **Design constraint requirements** are non-functional requireme
  are related to standards compliance and hardware limitations.

What is a feasibility study and what is its role?

- A feasibility study is a short focused study that checks if the syst
  contributes to organizational objectives the system can be engine
  current technology and within budget the system can be integrate
  other systems that are used.
- A feasibility study is used to decide if the proposed system is wo

- Feasibility studies can also help answer some of the following quest

1. What if the system wasn’t implemented?
2. What are current process problems?
3. How will the proposed system help?
4. What will be the integration problems?
5. Is new technology needed? What skills?
6. What facilities must be supported by the proposed system?

Requirements elicitation

- Requirements elicitation involves working with customers to dete application domain, the services that the system should provide, a operational constraints of the system.

- Elicitation may involve end-users, managers, engineers involved maintenance, domain experts, trade unions, and so on. These peop collectively called **stakeholders**.

- But stakeholders don’t always know what they really want.

- The software engineer has to be sensitive to the needs of the stakeho
  and aware of the problems that stakeholders can create including: ex
  requirements in their own terms
    - providing conflicting requirements
    - introducing organizational and political factors, which may influence the sy
      requirements
    - changing requirements during the analysis process due to new stakeholders
      emerge and changes to the business environment
- The software engineer must monitor and control these factors throu
  requirements engineering process.

The following three approaches to requirements elicitation will b discussed in detail:

- **Joint application design (JAD)**
- **Quality function deployment (QFD)**
- **Designer as apprentice**

JAD

- **Joint application design (**JAD) involves highly structured gro meetings or mini-retreats with system users, system owners, ar analysts in a single room for an extended period.
- These meetings occur four to eight hours per day and over a pe lasting one day to a couple of weeks.

Software engineers can use JAD for:

i. eliciting requirements and for the SRS

ii. design and software design description

iii. code

iv. tests and test plans

v. user manuals

- Planning for a review or audit session involve

three steps:

i. selecting participants

ii. preparing the agenda

iii. selecting a location

Rules for JAD sessions

The session leader must make every effort to ensure that these practices are implem

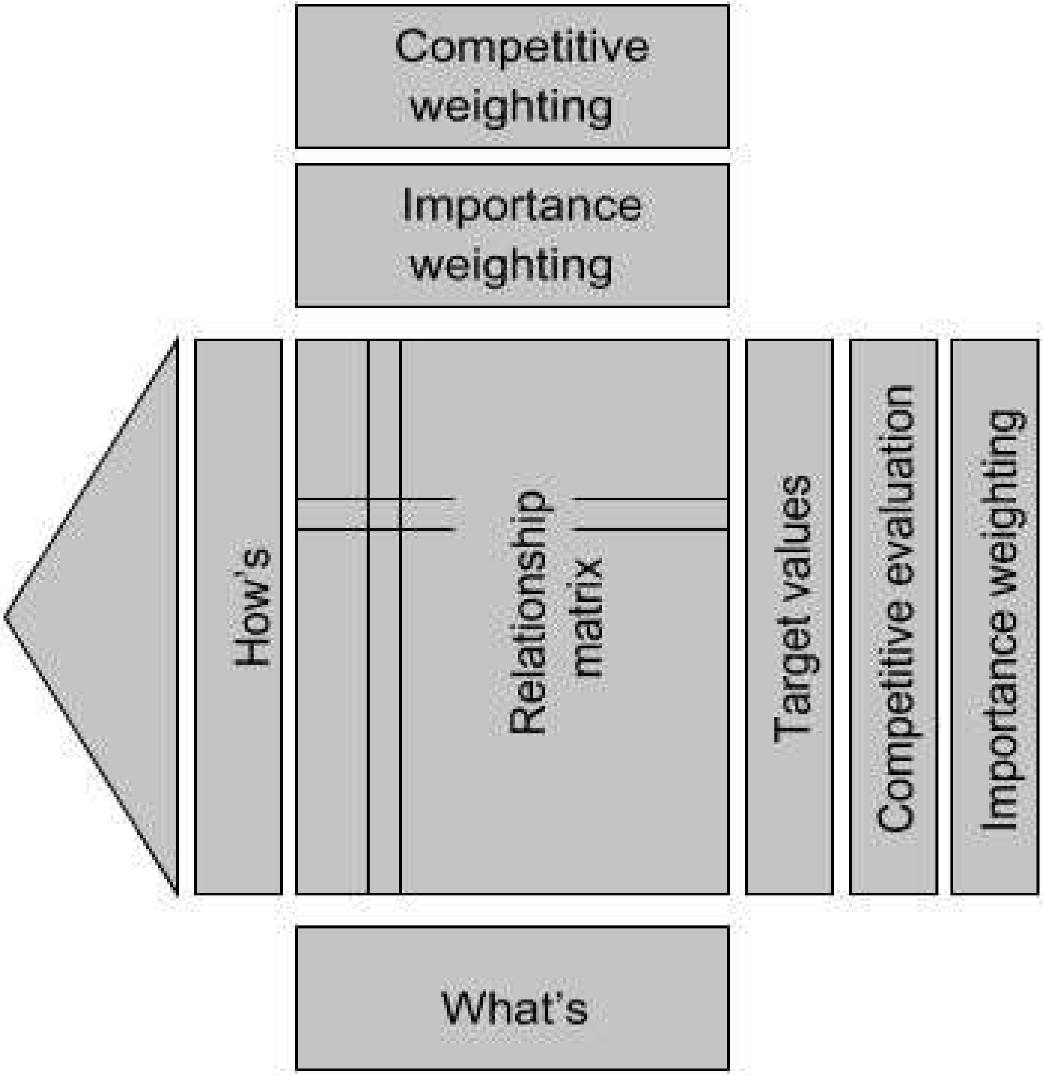
- Stick to the agenda.
- Stay on schedule
- Ensure that the scribe is able to take notes.
- Avoid technical jargon.
- Resolve conflicts.
- Encourage group consensus.
- Encourage user and management participation without allowing individuals t dominate the session.
- Keep the meeting impersonal.

Quality function deployment (QFD)

- It is a technique for determining customer requirements and defi
major quality assurance points to be used through-out the produc
phase.

- QFD provides a structure for ensuring that customers’ wants and
are carefully heard, and then directly translated into a company’s
internal technical requirements from analysis through implemen
deployment.

- The basic idea of QFD is to construct relationship matrices betw customer needs, technical requirements, priorities, and competitor assessment.
- These relationship matrices are often represented as the roof, ce and sides of a house, QFD is sometimes referred to as the “hous quality”


Competitive weighting
Importance weighting
How's
Relationship matrix
Target values
Competitive evaluation
Importance weighting
What's

Advantages of QFD

- QFD improves the involvement of users and mangers.
- It shortens the development life cycle and improves overall pro development.
- QFD supports team involvement by structuring communication processes.
- It provides a preventive tool to avoid the loss of information.

Designer as apprentice

- Designer as apprentice is a requirements discovery technique in which the requirem
  “looks over the shoulder” of the customer to enable the engineer to learn enough ab
  of the customer to understand
  his needs.

- The relationship between customer and designer is like that between a master crafts
  apprentice.

- The apprentice learns a skill from the master just as we want the requirements engir
  designer) to learn about the work from the customer.

- The apprentice is there to learn whatever the master knows.

- The designer must understand the structure and implication of the work, including

  a. the strategy to get work done

  b. constraints that get in the way

  c. the structure of the physical environment as it supports work

  d. the way work is divided

  e. recurring patterns of activity

  f. the implications the above has on any potential system

- **Both customer and designer learn during this process; the customer learns wh possible and the designer expands his understanding of the work.**

- **Requirements modeling** involves the techniques needed to express requirements in
  capture user needs.
- There are a number of ways to model software requirements; these include natural la
  informal and semiformal techniques, user stories, use case diagrams, structured diag
  object-oriented techniques, formal methods, and more.
- English, or any other natural language, have many problems for requirements comm
- These problems include lack of clarity and precision, mixing of functional and non-f
  requirements, ambiguity, overflexibility, and lack of modularization.
- Every clear SRS must have a great deal of narrative in clear and concise natural lang
  when it comes to expressing complex behavior, it is best to use formal or semiforma
  clear diagrams or tables, and narrative as needed to tie these elements together.

Use Cases

- Use cases are an essential artifact in object-oriented requirements
  elicitation and analysis and are described graphically using any o
  techniques.
- Artifact – something observed in a scientific investigation or exp
  that is not naturally present but occurs as a result of the preparati
  investigative procedure.
- One representation for the use case is the use case diagram, whic
  the interactions of the software system with its external environm

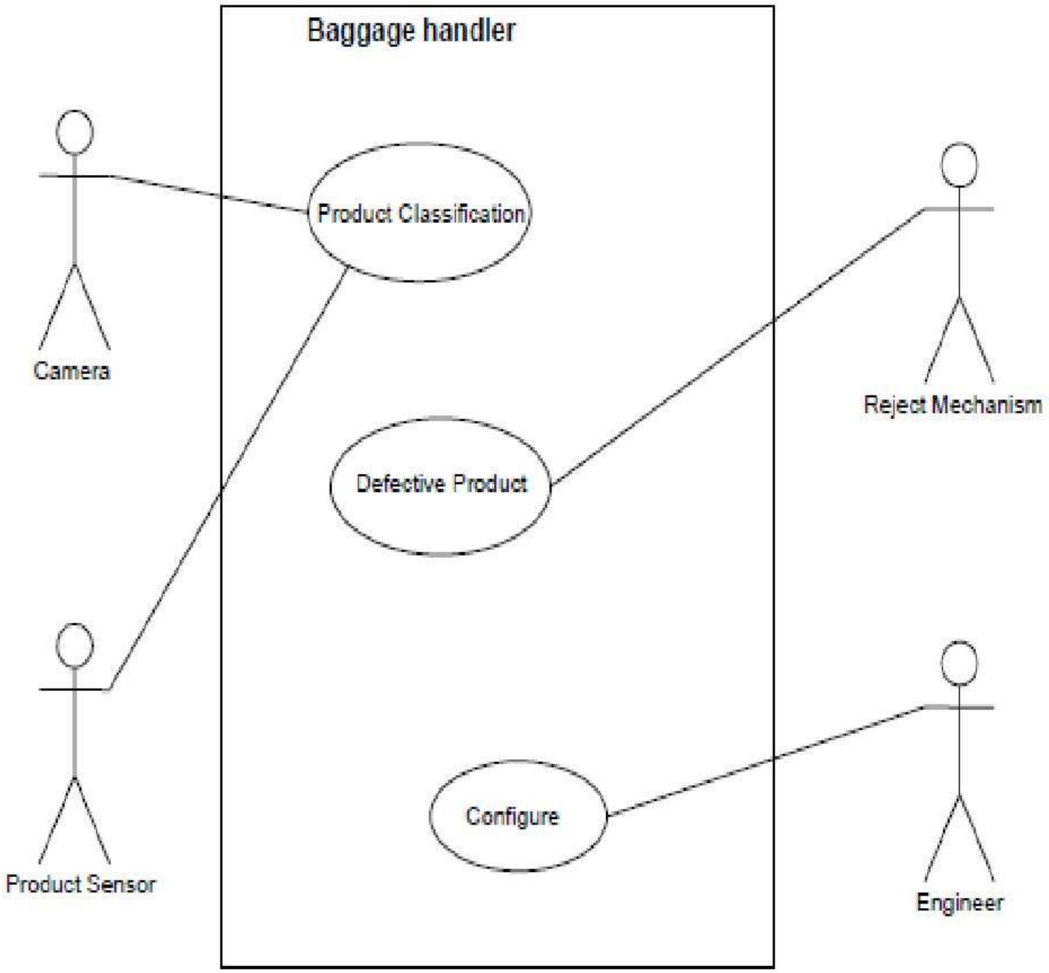


FIGURE 3.3
Use case diagram of the baggage inspection system.

In a use case diagram, the b
represents the system itself.
The stick figures represent “
designate external entities th
with the system.
The actors can be humans, o
systems, or device inputs.
Internal ellipses represent ea
of use for each of the actors
The solid lines associate act
each use.

- Each use case is a document that describes scenarios of operation system under consideration as well as pre- and post conditions an exceptions.
- In an iterative development life cycle, these use cases will becom increasingly refined and detailed as the analysis and design work progress.

User Stories

- User stories are short conversational texts that are used for initial requ discovery and project planning.
- User stories are widely used in conjunction with agile methodologies.
- User stories are written by the customers in their own “voice,” in term the system needs to do for them.
- User stories usually consist of two to four sentences written by the cu his own terminology, usually on a three-by-five inch card.

- The appropriate amount of user stories for one system increment or evolutic
  80, but the appropriate number will vary widely depending upon the applica
  and scope
- User stories should provide only enough detail to make a reasonably low ris
  of how long the story will take to implement.
- When the time comes to implement the story, developers will meet with the
  to flesh out the details.
- User stories also form the basis of acceptance testing.
  - For example, one or more automated acceptance tests can be created to
    user story has been correctly implemented.

Requirements Documentation

- The SRS document is the official statement of what is required of the systen developers.
- The SRS should include both a definition and a specification of requirement
- SRS is *not* a design document. It should be a set of *what* the system should c than *how* it should do it.
- A variety of stakeholders uses the software requirements throughout the soft cycle.

- Stakeholders include customers (these might be external custo
or internal customers such as the marketing department), mana
developers, testers, and those who maintain the system.
- Each stakeholder has a different perspective on and use for the

Software Stakeholders and Their Uses of the SRS

Stakeholder	Use
Customers	Express how their needs can be met. They continue to do this throughout the process as their perceptions of their own needs change.
Managers	Bid on the system and then control the software production process.
Developers	Create a software design that will meet the requirements.
Test engineers	A basis for verifying that the system performs as required.
Maintenance engineers	Understand what the system was intended to do as it evolves over time.

- **Requirements traceability** is concerned with the relationships betw
requirements, their sources, and the system design.
- During the requirements engineering process, the requirements must
identified to assist in planning for the many changes that will occur th
the software life cycle.
- **Requirements engineering** is the process of eliciting, documenting,
validating, and managing requirements.
- Traceability is also an important factor in conducting an impact analy
determine the effort needed to make such changes.

Characteristics of good requirements

Clear

- Good requirements are clear, concise, and easy to understand.
- That means they can’t be pumped full of management-speak, and confusing
- It is okay to use technical terms and abbreviations if they are defined some they are common knowledge in the project's domain.
- In short, requirements cannot be vague or ill-defined.
- Each requirement must state in concrete

Unambiguous

- A requirement must be unambiguous.
- If the requirement is worded so that we can’t tell what it requires, then we ca
  system to satisfy it.
- Although this may seem like an obvious feature of any good requirement, it’s
  sometimes harder to guarantee than we might think.
- As we write requirements, do our best to make them unambiguous.
- Read them carefully to make sure we can’t think of any way to interpret them
  than the way we intend.

Consistent

- A project’s requirements must be consistent with each other
- That means not only that they cannot contradict each other, that they also don’t provide so many constraints that the problem is unsolvable.
- Each requirement must also be self-consistent.(In other wo it must be possible to achieve.)

Prioritized

- When we start working on the project’s schedule, it’s likely we need to c few nice-to-haves from the design.
- We might like to include every feature but don’t have the time or budget something’s got to go.
- At this point, we need to prioritize the requirements.
- If we have assigned costs (usually in terms of time to implement) and pr to the requirements, then we can defer the high-cost, low-priority requirements until a later release.

Verifiable

- Requirements must be verifiable.
- If we can’t verify a requirement, how do we know whether we met it?
- Being verifiable means the requirements must be limited and p defined.

The characteristics of good requirements. They are as follows:

- **Correct** — The SRS should correctly describe the system behavior.
- **Unambiguous** — An unambiguous SRS is one that is clear and not subject to different interpretations.
  appropriate language can help avoid ambiguity.
- **Complete** — An SRS is complete if it completely describes the desired behavior.
- **Consistent** — One requirement must not contradict another
- **Ranked** — An SRS must be ranked for importance and/or stability. Not every requirement is as critica
  By ranking the requirements, designers will find guidance in making tradeoff decisions.
- **Verifiable** — Any requirement that cannot be verified is a requirement that cannot be shown to have be
- **Modifiable** — The requirements need to be written in such a way so as to be easy to change.
- **Traceable** — The SRS must be traceable because the requirements provide the starting point for the tra
  chain.

REQUIREMENT CATEGORIES

Audience-Oriented Requirements:

- These categories focus on different audiences and the different points of view audience has.
- They use a somewhat business-oriented perspective to classify requirements to the people who care the most about them.

Business Requirements:

- Business requirements layout the project’s high-level goals. They explain wh customer hopes to achieve with the project.

User Requirements:

- User requirements (which are also called stake holder requirements), descr

  the project will be used by the end users.

- They often include things like sketches of forms, scripts that’s how the step

  will perform to accomplish specific tasks, use cases, and prototypes.

Functional Requirements:

- Functional requirements are detailed statements of the project’s desired cap
- They are similar to the user requirements but they may also include things

  users won’t see directly.

Non-functional Requirements:

- Non-functional requirements are statements about the quality of the applica
  behavior or constraints on how it produces a desired result.
- They specify things such as the application’s performance, reliability, and s
  characteristics.

Implementation Requirements:

- Implementation requirements are temporary features that are needed to tran
  using the new system but that will be later discarded.
- After we finish testing the system and are ready to use it full time, you need
  copy any pending invoices from the old database into the new one.

GATHERING REQUIREMENTS

- **Listen to Customers (and Users):**
  - Start by listening to the customers.
  - Learn as much as we can, about the problem they are trying to address and
- **Brain Storming** – is a group activity exercise which helps to develo solutions.
- **Use the Five Ws (and One H):**
  - **Who :** ask who will be using the software and get to know as much as we c those people

- **What :-** Figure out what the customers need the application to do.
- **When :-** Find out when the application is needed.
- **Where :-** Find out where the application will be used. Will it be us desktop computers in an air-conditioned office? Or will it be used in a noisy subway?
- **Why :-** Ask why the customers need the application. Use the “why to help clarify the customers’ needs and see if it is real.
- **How :-**We shouldn’t completely ignore the customers’ ideas. Some customers have good ideas