srs

- Requirements engineering concepts
- Requirements specifications
- Requirements elicitation
- Requirements modeling
- Requirements documentation
- Recommendations on requirements

- Requirements engineering is the process of eliciting, documenting, analysing, validating, and managing requirements.

- Requirements modeling involves the techniques needed to express requirements in a way that can capture user needs.

- Requirements modeling uses techniques that can range from high-level abstract statements through psuedocode-like specifications, formal logics, and graphical representations.

- The requirements engineer must always strive to gather complete, precise, and detailed specifications of system requirements.

- **What is software requirements engineering?**
- Requirements engineering is a sub discipline of software engineering that is concerned with determining the goals, functions, and constraints of software systems.
- Requirements engineering also involves the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.

When does requirements engineering start?

- the requirements engineering process begins with a feasibility study activity, which leads to a feasibility report.
- It is possible that the feasibility study may lead to a decision not to continue with the development of the software product.
- If the feasibility study suggests that the product should be developed, then requirements analysis can begin.

- **What is software requirements specification?**
- This is the set of activities designed to capture behavioral and nonbehavioral aspects of the system in the SRS document.
- The goal of the SRS activity, and the resultant documentation, is to provide a complete description of the system's behaviour without describing the internal structure.

- **Why do we need SRSs?**

- Precise software specifications provide the basis for analyzing the requirements, validating that they are the stakeholder’s intentions, defining what the designers have to build, and verifying that they have done so correctly.

- **How do software requirements help software engineers?**
- SRSs allow us to know the motivation for development of the software system.
- Software requirements also help software engineers manage the evolution of the software over time and across families of related software products.
- This approach reflects the reality of a changing world and the need to reuse partial specifications.

- **What are the core requirements engineering activities?**
- First, we must elicit the requirements, which is a form of discovery.
- Some people use the term “gathering” to describe the process of collecting software requirements, but “gathering” implies that requirements are like vegetables in the garden to be harvested.
- Often, though, requirements are deeply hidden, expressed incorrectly by stakeholders, contradictory, and complex.
- Eliciting requirements is one of the hardest jobs for the requirements engineer.

- Modeling requirements involves representing the requirements in some form.

- Words, pictures, and mathematical formulas can all be used but it is never easy to effectively model requirements.

- Analyzing requirements involves determining if the requirements are correct or have certain other properties such as consistency, completeness, sufficient detail, and so on.

- The requirements models and their properties must also be communicated to stakeholders and the differences reconciled.

- Finally, requirements change all the time and the requirements engineer must be equipped to deal with this eventuality. We will focus our discussions on these aforementioned areas

- **Which disciplines does requirements engineering draw upon?**
- Requirements engineering is strongly influenced by computer science and systems engineering.

- But because software engineering is a human endeavor, particularly with respect to understanding people's needs, requirements engineering draws upon such diverse disciplines as philosophy, cognitive psychology, anthropology, sociology, and linguistics.

- **What is a requirement?**
- A requirement can range from a high-level, abstract statement of a service or constraint to a detailed, formal specification.

- There is so much variability in requirements detail because of the many purposes that the requirements must serve.

Requirements Specifications

- **What kinds of SRSs are there?**
- SRSs are usually classified in terms of their level of abstraction:
- • user requirements
- • system requirements
- • software design specifications
- **What are user requirements specifications?**
- User requirements specifications are usually used to attract bidders, often in the form of a request for proposal (RFP).

- User requirements may contain abstract statements in natural language, for example, English, with accompanying informal diagrams, even back-of- the-napkin drawings*.

- User requirements specify functional and nonfunctional requirements as they pertain to externally visible behavior in a form understandable by clients and system users.

- **What are system requirements specifications?**

- System level requirements, or SRSs mentioned previously, are detailed descriptions of the services and constraints.

- Systems requirements are derived from analysis of the user requirements.

- Systems requirements should be structured and precise because they act as a contract between client and contractor (and can literally be enforced in court).

What are software design specifications?

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

- **Within these three specification types are there different requirements types?**
- Yes, there are. Within the family of user, system, and functional requirements specifications, all kinds of things can be described.

- These include external constraints to the software and user needs.

- The user needs are usually called “functional requirements” and the external constraints are called “non- functional requirements.”

- **What are functional requirements?**
- Functional requirements describe the services the system should provide.
- Sometimes the functional requirements state what the system should not do.
- Functional requirements can be high-level and general or detailed, expressing inputs, outputs, exceptions, and so on.

- **What are nonfunctional requirements?**

- Nonfunctional requirements are imposed by the environment in which the system is to exist.

- These requirements could include timing constraints, quality properties, standard adherence, programming languages to be used,
- compliance with laws, and so on.

What are domain requirements?

- Domain requirements are a type of nonfunctional requirement from which the application domain dictates or derives.
- Domain requirements might impose new functional requirements or constraints on existing functional requirements.
- For example, in the baggage inspection system, industry standards and restrictions on baggage size and shape will place certain constraints on the system.

- **What are interface specifications?**

- Interface specifications are functional software requirements specified in terms of interfaces to operating units.

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

- There are three types of interface that may have to be defined:
- 1. procedural interfaces
- 2. data structures that are exchanged
- 3. data representations
- Formal notations are an effective technique for interface specification

- **What are performance requirements?**

- Performance requirements are functional requirements that include the static and dynamic numerical requirements placed on the software or on human interaction with the software as a whole.
- For an imaging system, static requirements might include the number of simultaneous users to be supported.
- The dynamic requirements might include the numbers of transactions and tasks the amount of data to be processed within certain time periods for both normal and peak workload conditions.

- **What are logical database requirements?**
- Logical database requirements are functional requirements that include the types of information used by various functions such as frequency of use, accessing capabilities, data entities and their relationships, integrity constraints, and data retention requirements.
- **What are design constraint requirements?**
- Design constraint requirements are non functional requirements that are related to standards compliance and hardware limitations.

- **What are system attribute requirements?**

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

- **What is a feasibility study and what is its role?**
- A feasibility study is a short focused study that checks if the system contributes to organizational objectives the system can be engineered using current technology and within budget the system can be integrated with other systems that are used.
- A feasibility study is used to decide if the proposed system is worthwhile.
- Feasibility studies can also help answer some of the following questions.
- What if the system wasn't implemented?
- What are current process problems?
- How will the proposed system help?
- What will be the integration problems?
- Is new technology needed? What skills?
- What facilities must be supported by the proposed system?
- Therefore, it is important to conduct a feasibility study before a large investment is made in a system that is not needed or which does not solve some inherent business process problem.

- **Are there social and organizational factors in requirements engineering?**
- You bet there are. Software engineering involves many human activities, and none is more obviously so than requirements engineering.
- Software systems are used in a social and organizational context; thus, political factors almost always influence the software requirements.
- Moreover, stakeholders often don’t know what they really want and they express requirements in their own terms.
- Different stakeholders may have conflicting requirements, new stakeholders may emerge, and the business environment may change.
- Finally, the requirements change during the analysis process for a host of reasons.
- Therefore, good requirements engineers must be sensitive

requirements elicitation

What is requirements elicitation?

- Requirements elicitation involves working with customers to determine the application domain, the services that the system should provide, and the operational constraints of the system.
- Elicitation may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, and so on. These people are collectively called stakeholders.

But stakeholders don’t always know what they want, right?

- An important consideration in eliciting requirements from stakeholders is that they often don’t know what they really want
- The software engineer has to be sensitive to the needs of the stakeholders and aware of the problems that stakeholders can create including:
- **expressing requirements in their own terms**
- **providing conflicting requirements**
- **introducing organizational and political factors, which may influence the system requirements**
- **changing requirements during the analysis process due to new stakeholders who may emerge and changes to the business environment** .
- The software engineer must monitor and control these factors throughout the requirements engineering process.

Are there any practical approaches to requirements elicitation?

Yes, there are several.

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

Each of these techniques has been used successfully for requirements elicitation**(assignment 1)**

Requirements Modeling

How are software requirements modeled?

- There are a number of ways to model software requirements
- natural languages, informal and semiformal techniques, user stories, use case diagrams, structured diagrams, object-oriented techniques, formal methods, and more.

Why can't requirements just be communicated in English?

- English, or any other natural language, is fraught with problems for requirements
- communication.
- These problems include lack of clarity and precision, mixing of functional and nonfunctional requirements, and requirements amalgamation, where several different requirements may be expressed together.
- Other problems with natural languages include ambiguity, overflexibility, and lack of modularization.
- These shortcomings, however, do not mean that natural language is never used in an SRS.
- Every clear SRS must have a great deal of narrative in clear and concise natural language.
- But when it comes to expressing complex behavior, it is best to use formal or semiformal methods, clear diagrams or tables, and narrative as needed to tie these elements together.

What are some alternatives to using natural languages?

Alternatives to natural languages include:

- structured natural language
- design description languages
- graphical notations
- mathematical specifications

What is a structured language specification?

- This is a limited form of natural language that can be used to express requirements.
- In other words, the vocabulary and grammar rules available to express requirements are strictly controlled and are capable of being parsed.
- Structured languages remove some of the problems resulting from ambiguity and flexibility and impose a degree of uniformity on a specification.
- use of structured languages requires a level of training that can frustrate stakeholders.
- To increase usability, structured languages can be facilitated using **a forms-based approach.**
- In a forms-based approach, templates or forms are created and given to the customer and other stakeholders to be filled in.
- The template can include the following information:

definition of the function or entity
description of inputs and where they come from
description of outputs and where they go to
indication of other entities required
pre- and postconditions (if appropriate)
side effects (if any)

- The data in the template can then be converted to structured language using an appropriate interpreter.

Function	
Description	
Inputs	
Outputs	
Destination	
Requires	
Precondition	
Postcondition	
Side-effects	

FIGURE 3.2
A forms-based specification template.

What are program design language-based requirements?

- Program design languages (PDLs) involve requirements that are defined operationally using a language like a programming language but with more flexibility of expression.

PDLs are most appropriate in two situations:

1. where an operation is specified as a sequence of actions and the order is important
2. when hardware and software interfaces have to be specified

Are there any disadvantages to using PDLs?

- the PDL may not be sufficiently expressive to define domain concepts.
- the specification may be taken as a design rather than a specification.
- the notation may be understandable only to people with knowledge of programming languages.

What are use cases?

- Use cases are an essential artifact in object-oriented requirements elicitation and analysis and are described graphically using any of several techniques.
- One representation for the use case is the use case diagram, which depicts the interactions of the software system with its external environment.
- In a use case diagram, the box represents the system itself.
- The stick figures represent “actors” that designate external entities that interact with the system.
- The actors can be humans, other systems, or device inputs.
- Internal ellipses represent each activity of use for each of the actors (use cases.)
- The solid lines associate actors with each use.

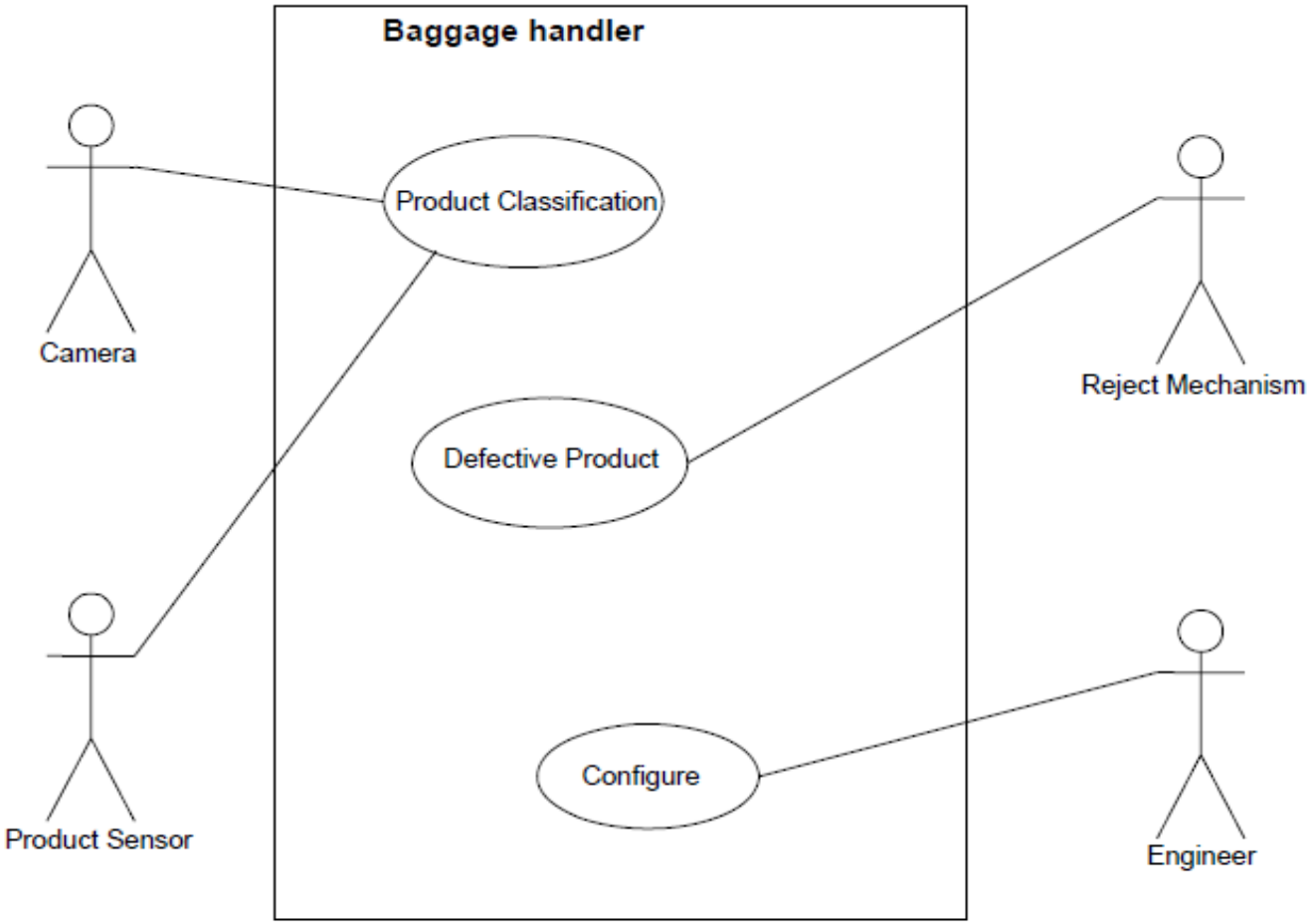

FIGURE 3.3
Use case diagram of the baggage inspection system.

- Each use case is, a document that describes scenarios of operation of the system under consideration as well as pre- and post conditions and exceptions.
- In an iterative development life cycle, these use cases will become increasingly refined and detailed as the analysis and design workflows progress.
- Interaction diagrams are then created to describe the behaviors defined by each use case.
- In the first iteration these diagrams depict the system as a “black box,” but once domain modeling has been completed the black box is transformed into a collaboration of objects as will be seen later.

What are user stories?

- User stories are short conversational texts that are used for initial requirements 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 terms of what the system needs to do for them.
- User stories usually consist of two to four sentences written by the customer in his own terminology, usually on a three-by-five inch card.
- The appropriate amount of user stories for one system increment or evolution is about 80, but the appropriate number will vary widely depending upon the application size and
- scope.

- User stories should provide only enough detail to make a reasonably low risk estimate of how long the story will take to implement.
- When the time comes to implement the story, developers will meet with the customer 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 verify the user story has been correctly implemented.

What are formal methods in software specification?

- Formal methods attempt to improve requirements formulation and expression by applying mathematics and logic.
- Formal methods employ some combination of predicate calculus (first order logic), recursive function theory, Lambda calculus, programming language semantics, discrete mathematics, number theory, and abstract algebra.
- This approach is attractive because it offers a more scientific method for requirements specification.
- By their nature, specifications for most embedded systems usually contain some formality in the mathematical expression of the underlying imaging operations

What are the motivations for using formal methods?

- One of the primary attractions of formal methods is that they offer a highly scientific approach to developmen.
- Formal requirements offer the possibility of discovering errors at the earliest phase of development, while the errors can be corrected quickly and at a low cost.
- Informal specifications might not achieve this goal because they are not precise enough to be refuted by finding counter examples.

What are informal and semiformal methods?

- Approaches to requirements specification that are not formal are either informal (such as flow-charting) or semiformal.

- The UML is a semiformal specification approach, meaning that while it does not appear to be mathematically based, it is in fact nearly formal in that every one of its modeling tools can be converted either completely or partially to an underlying mathematical
- representation (a work group is focused on remedying these deficiencies).

- In any case, UML largely enjoys the benefits of both informal and formal techniques.

How are formal methods used?

- Formal methods are typically not intended to take on an all-encompassing role in system or software development.
- Instead, individual techniques are designed to optimize one or two parts of the development life cycle.
- There are three general uses for formal methods:

1. Consistency checking — system behavioral requirements are described using a mathematically based notation.

2. Model checking — state machines are used to verify if a given property is satisfied under all conditions.

3. Theorem proving — axioms of system behavior are used to derive a proof that a system will behave in a given way.

Are formal methods hard to use?

- Formal methods can be difficult to use and are sometimes error-prone.
- For these reasons and because they are sometimes perceived to increase early life-cycle costs and delay projects, formal methods are frequently and unfortunately avoided.

What are some of the formal methods techniques?

Formal methods include Z, Vienna design method (VDM), and communicating sequential processes (CSP). All of these methods are highly specialized and require a great deal of formal mathematical training that most traditional engineers do not receive**.(assignment1)**

Requirements Documentation

What is the role of the SRS?

- The SRS document is the official statement of what is required of the system developers. The SRS should include both a definition and a specification of requirements.
- However, the SRS is *not* a design document.
- As much as possible, it should be a set of *what* the system should do rather than *how* it should do it.
- Unfortunately, the SRS usually contains some design specifications, which has the tendency to hamstring the designers.

Who uses the requirements documents?

- A variety of stakeholders uses the software requirements throughout the software life cycle.
- Stakeholders include customers (these might be external customers or internal customers such as the marketing department), managers, developers, testers, and those who maintain the system.
- Each stakeholderhas a different perspective on and use for the SRS.

TABLE 3.5

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.

has a different perspective on and use for the SRS. Various stakeholders and their uses for the SRS are summarized in Table 3.5.

How do I organize the requirements document?

- There are many ways to organize the SRS.
- IEEE Standard 830-1998 Recommended Practice for Software Requirements Specifications (SRS) provides a template of what an SRS should look like.

- The SRS starts with “boilerplate” front matter, which usually acts as a preamble to the product.
- For example, in Appendix A, most of Sections 1 and 2 introduce the wet well control system terminology and operating environment.

- Following the boilerplate material, the functionality of the system is described in one or more styles (or “views”).
- IEEE 830 recommends that the views include some combination of decomposition, dependency, interface, and detail descriptions.
- Together with the boilerplate front matter, these form a standard template for SRSs, which is depicted in fig 3.15
- An outline of some specific requirements, or work breakdown structure (WBS), for the baggage inspection system, in IEEE 830 format, is given in Figure 3.16.

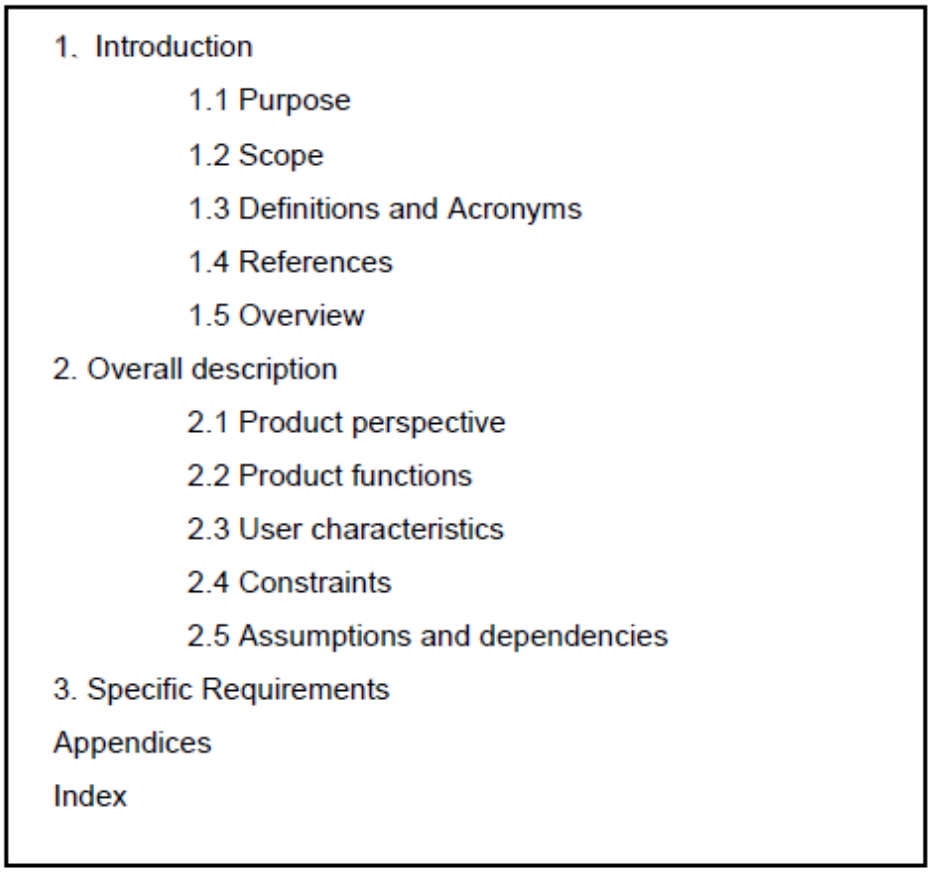

FIGURE 3.15
Recommended Table of Contents for an SRS from IEEE Standard 830–1998.

How do you represent specific requirements in the SRS?

- The IEEE 830 standard provides for several alternative means to represent
- the requirements specifications, aside from a function perspective.
- In particular, the software requi ements can be organized by
- **functional mode (“operational,” “diagnostic,” “calibration”)**
- **user class (“operator,” “diagnostic”)**
- **object**
- **feature (what the system provides to the user) • stimulus (sensor 1, 2, etc.)**
- **functional hierarchy**
- **mixed (a combination of two or more of the above**

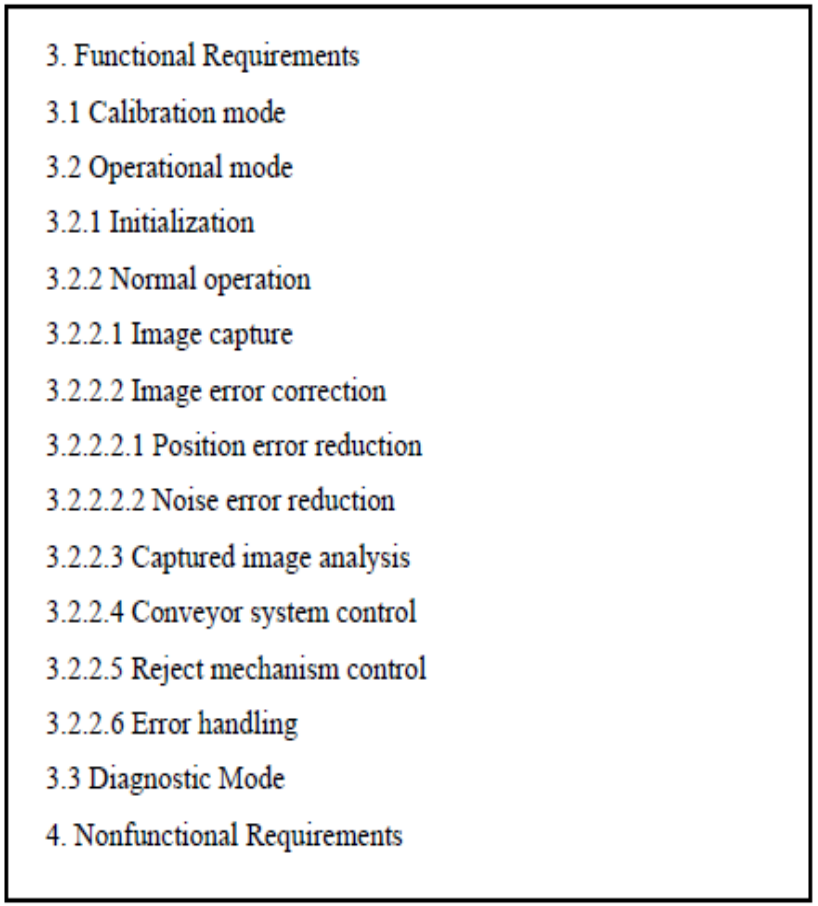

FIGURE 3.16
Some specific requirements for the baggage inspection system.

What is requirements traceability?

- Requirements traceability is concerned with the relationships between requirements, their sources, and the system design.
- Requirements can be linked to the source, to other requirements, and to design elements.
- Source traceability links requirements to the stakeholders who proposed these requirements.
- Requirements traceability links between dependent requirements. Design traceability links from the requirements to the design.

What role does traceability help in requirements documentation?

- During the requirements engineering process, the requirements must be identified to assist in planning for the many changes that will occur throughout the software life cycle.
- Traceability is also an important factor in conducting an impact analysis to determine the effort needed to make such changes.

What are traceability policies?

- Traceability policies determine the amount of information about requirements relationships that need to be maintained.
- There are a number of open source and proprietary CASE tools that can help improve requirements traceability.

What does a traceability matrix look like?

- One type of traceability matrix is shown in Table 3.6. Requirements identification numbers label both the rows and columns.
- **An "R" is placed in a corresponding cell if the requirement in that row references the requirement in that column.**
- **A "U" corresponds to an actual use dependency between the two requirements.**

TABLE 3.6

A Sample Traceability Matrix

Requirement Identification	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

Recommendations on Requirements

Is there a preferred modeling technique for an SRS?

- It is risky to prescribe a preferred technique because it is well known that there is no "silver bullet" when it comes to software specification and design and each system should be considered on its own merits.
- Nevertheless, regardless of approach, any SRS should incorporate the following best practices:

• Use consistent modeling approaches and techniques throughout the specification; for example, top-down decomposition, SA or OOA.

• Separate operational specification from descriptive behavior.

• Use consistent levels of abstraction within models and conformance between levels of refinement across models.

• Model nonfunctional requirements as a part of the specification models, in particular timing properties.

• Omit hardware and software assignment in the specification (another aspect of design rather than specification).

Are there special challenges when engineers specify software systems?

The challenges include:

- Mixing of operational and descriptive specifications.
- Combining low-level hardware functionality and high-level systems and software functionality in the same functional level.
- Omission of timing information.

the IEEE 830 standard describes **the characteristics of good requirements**. They are as follows:

Correct — The SRS should correctly describe the system behavior. Obviously, it is unhelpful if the SRS incorrectly defines the system or somehow involves unreasonable expectations such as defying the laws of physics.

Unambiguous — An unambiguous SRS is one that is clear and not subject to different interpretations. Using appropriate language can help avoid ambiguity.

Complete — An SRS is complete if it completely describes the desired behavior. Ordinarily, the note “TBD,” that is “to be defined (later)” is unacceptable in a requirements document. IEEE 830 sets out some exceptions to this rule.

Consistent — One requirement must not contradict anotherRanked — An SRS must be ranked for importance and/or stability. Not every requirement is as critical as another. 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 been met.
Modifiable — The requirements need to be written in such a way so as to be easy to change. In many ways, this approach is similar to the information hiding principle.
Traceable — The SRS must be traceable because the requirements provide the starting point for the traceability chain. Approaches to traceability and its benefits have been mentioned at length

What wording is appropriate in requirements specifications?

To meet these criteria and to write clear requirements documentation, there are several best practices that the requirements engineer can follow.

They are as follows:

- Invent and use a standard format for all requirements.
- Use language in a consistent way.
- Use “shall” for mandatory requirements. • Use “should” for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of technical language unless it is warranted.

How do I recognize bad requirements?

To illustrate, consider the following bad requirements.

- “The systems shall be completely reliable.”
- “The system shall be modular.”
- “The system shall be maintainable.”
- “The system will be fast.”
- “Errors shall be less than 99%.”

What is requirements triage?

The following is good summary advice from requirements engineering specialist Al Davis. When dealing with the swirl of issues involved with requirements engineering, he suggests that you:

- Maintain a list of requirements.
- Record necessary interdependencies.
- Annotate requirements by effort.
- Annotate requirements by relative importance.
- Perform triage overtly (involve stakeholders)
- Customers
- Developers
- Financial representatives.
- Base decisions on more than mechanics.
- Establish a teamwork mentality.
- Manage by probabilities of completion not absolutes.
- Understand the optimistic, pessimistic, and realistic approaches.
- Plan more than one release at a time.
- Replan before every new release.
- Don’t be intimidated into a solution.
- Find a solution before you proceed.
- Remember that prediction is impossible [Davis 2003].

What is requirements validation?

Requirements validation is tantamount to asking the question “Am I building the right software?” Too often, software engineers deliver a system that conforms to the SRS only to discover that it is not what the customer really wanted.

Requirements error costs are high, and therefore validation is very important.

Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

How are requirements validated?

There are number of ways of checking SRSs for conformance to the IEEE 830 best practices and for validity. These approaches include:

- requirements reviews
- systematic manual analysis of the requirements
- prototyping
- using an executable model of the system to check requirements
- test-case generation
- developing tests for requirements to check testability
- automated consistency analysis
- checking the consistency of a structured requirements description

Are there tools out there already?

There is a free downloadable tool from NASA, the Automated Requirement Measurement (ARM) tool [NASA 2006]. This tool measures the “goodness” of a requirements specification on two levels, macro and micro. The macro level indicators include:

- Size of requirements
- Text structure
- Specification depth
- Readability

How are reading statistics useful in assessing an SRS?

The readability statistics rely on traditional measures of human writing level such as:

- Flesch Reading Ease Index — the number of syllables per word and words per sentence.
- Flesch-Kincaid Grade Level Index — Flesch score converted to a grade level (standard writing is about 7th or 8th grade).
- Coleman-Liau Grade Level Index — uses word length in characters and sentence length in words to determine grade level.
- Bormuth Grade Level Index — also uses word length in characters and sentence length in words to determine grade level.