# Understanding Requirements

# What is a requirement?
(as per Sommerville)

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- This is because requirements serve dual function
  - May be the basis for a bid for a contract - therefore must be open to interpretation;
  - May be the basis for the contract itself - therefore must be defined in detail;
  - Both these statements may be called requirements.

# Requirements Engineering Overview

- Inception—ask a set of questions that establish …
  - 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 customer and the developer
- Elicitation—elicit requirements from all stakeholders
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
- Negotiation—agree on a deliverable system that is realistic for developers and customers

# Requirements Engineering Overview (contd)

- Specification—can be any one (or more) of the following:
  - A written document
  - A set of graphical models
  - A formal mathematical model
  - A collection of user scenarios (use-cases)
  - A prototype
- Validation—a review mechanism that looks 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 or unrealistic (unachievable) requirements.
- Requirements management

# Inception

- Identify stakeholders
  - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- The first questions
  - 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?

# The Next Set of Questions

These questions enable the requirements engineer to gain a better understanding of the problem and allow 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

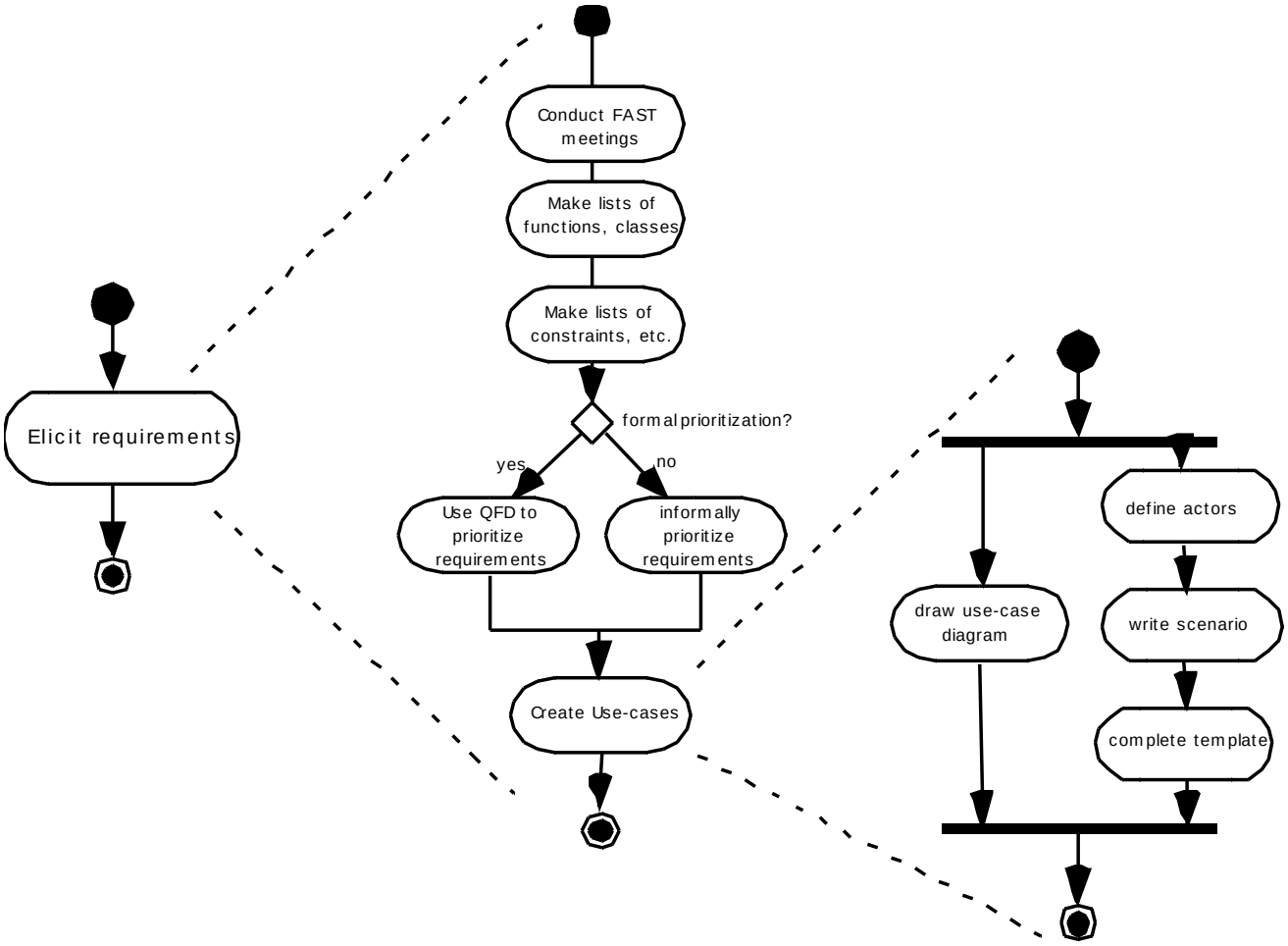
These questions focus on the effectiveness of the communication activity itself

- 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

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- an agenda is suggested
- 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
- the goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements

# Eliciting Requirements



# Quality Function Deployment

-Dr Yoji Akao et al (1966)

QFD is a technique that translates needs of customer into technical requirements for software

- Function deployment determines the "value" (as perceived by the customer) of each function required of the system

  A requirement may be

  - Normal - Stated
  - Expected - Implicit
  - Exciting - Beyond the above two
- Information deployment identifies data objects and events
- Task deployment examines the behavior of the system
- Value analysis determines the relative priority of requirements

# Elicitation Work Products

- 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.

# Bad example for Elicitation

# Building the Analysis Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

# SafeHome

The system would protect against and/or recognize a variety of undesirable situations such as illegal entry, fire, flooding, carbon monoxide level, and others. It will use wireless sensors to detect each situation.It can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

Participants in the discussion may

- –identify list of objects that are part of the environment, are produced, or used by the system to perform its functions.
- – identify list of services that manipulate, or interact with the objects.
- –prepare a list of constraints.

# SafeHome

Objects may include

- Control Panel
- Smoke detector
- Window and door sensors,
- Motion detectors,
- Display
- PC,
- Telephone Number
- Telephone Call
- etc.

# SafeHome

- Services may include
  - Configuring the System
  - Setting the Alarm
  - Monitoring the Sensors
  - Dialing the Phone
  - Programming the Control Panel
  - Reading the Display
- Constraints may include
  - Recognize working/non-working sensors
  - User friendly
  - Interface directly to a telephone
  - Performance criteria

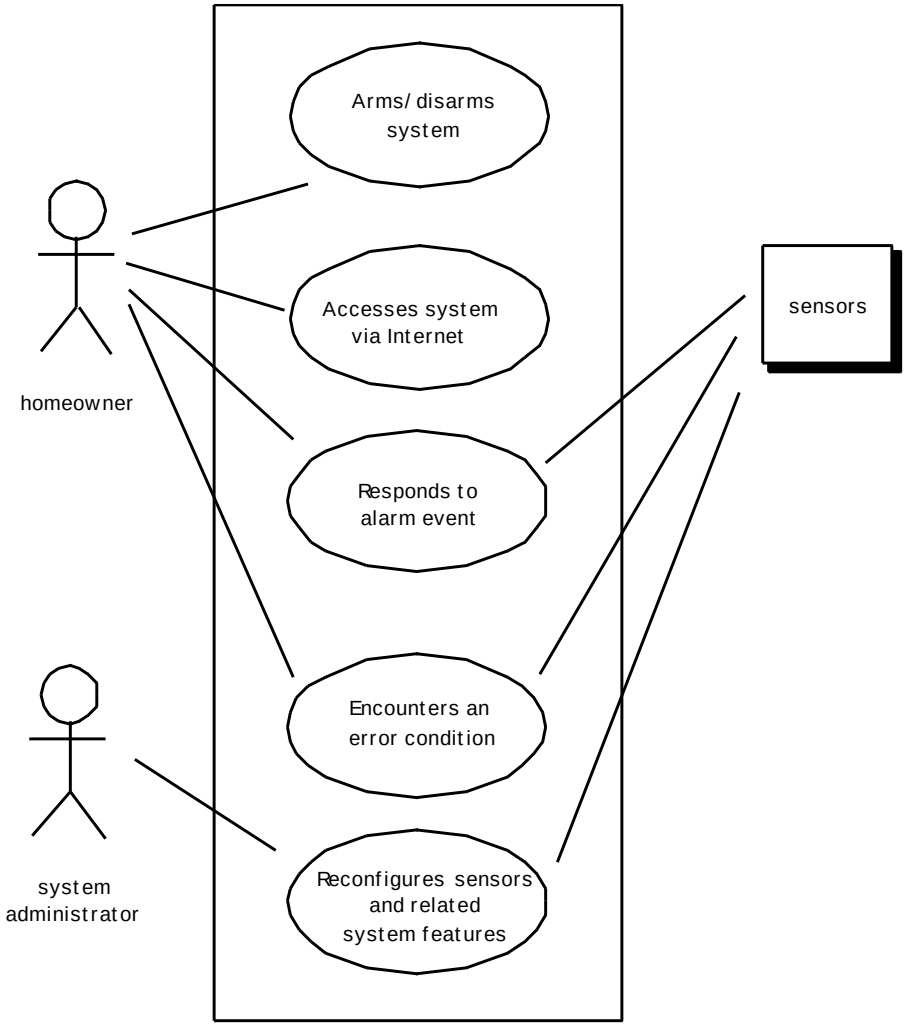
# Mini-spec for Control Panel

The control panel is a wall mounted unit that is approximately 9 X 5 inches in size. The control panel has wireless connectivity to sensors and a PC. User interaction occurs through a keypad containing 12 keys. A 3 X 3 inch LCD display provides user feedback. Software provides interactive prompts, echo, and similar functions.

# Use-Cases

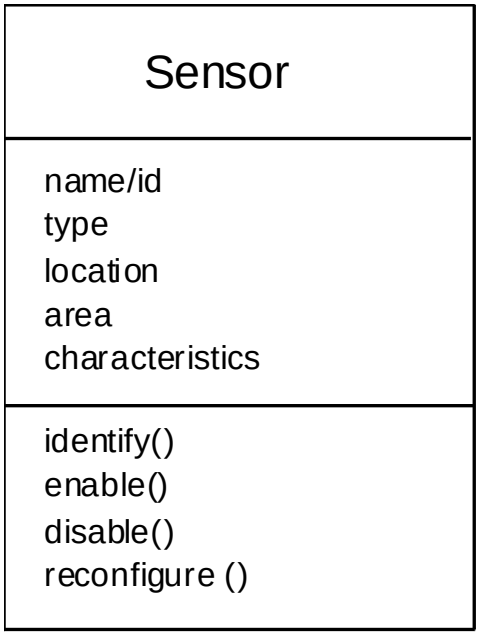
- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

# Use-Case Diagram

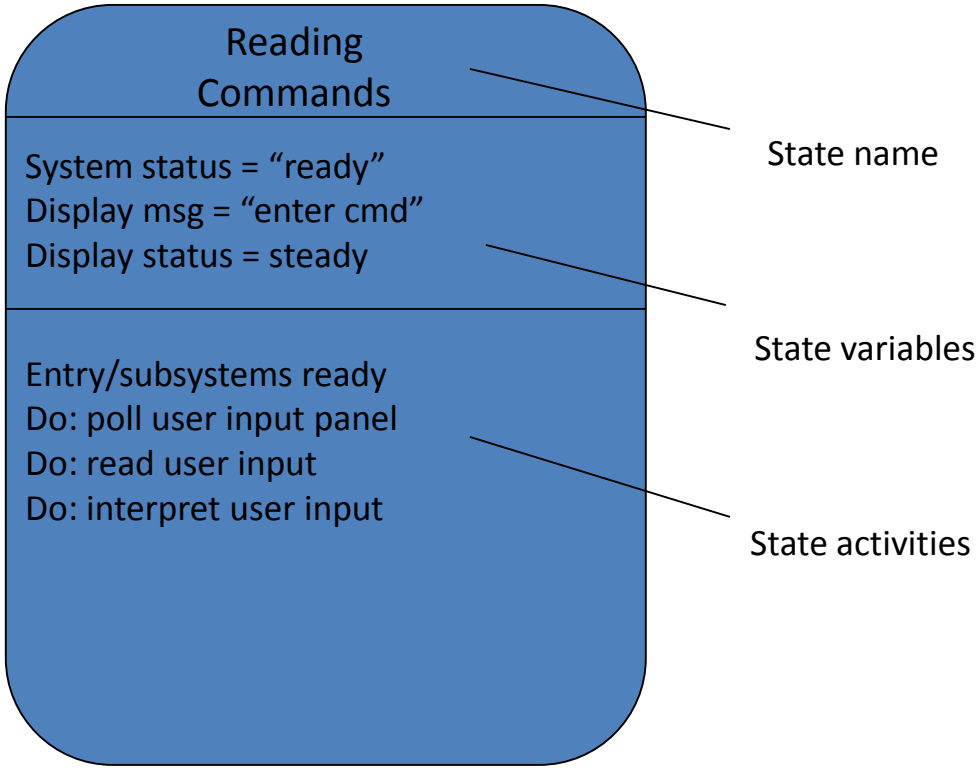


# Class Diagram

**From the *SafeHome* system …**



# State Diagram



# Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders "win conditions"
  - Win conditions are not always obvious
- Negotiate
  - Work toward a set of requirements that lead to "win-win"

# Problems with Requirements Practices

- We have trouble understanding the requirements that we do acquire from the customer
- We often record requirements in a disorganized manner
- We spend far too little time verifying what we do record
- We allow change to control us, rather than establishing mechanisms to control change
- Most importantly, we fail to establish a solid foundation for the system or software that the user wants built

The seeds of most software disasters are usually sown in first 3 months of commencing the software project
-Capers Jones

# Problems with Requirements Practices

(continued)

- Many software developers argue that
  - Building software is so compelling that we want to jump right in (before having a clear understanding of what is needed)
  - Things will become clear as we build the software
  - Project stakeholders will be able to better understand what they need only after examining early iterations of the software
  - Things change so rapidly that requirements engineering is a waste of time
  - The bottom line is producing a working program and that all else is secondary
- All of these arguments contain some truth, especially for small projects that take less than one month to complete
- However, as software grows in size and complexity, these arguments begin to break down and can lead to a failed software project

# Validating Requirements - I

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?

# Validating Requirements - II

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

# Functional and non-functional requirements
(as per Sommerville)

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.

- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.

- Domain requirements
  - Constraints on the system from the domain of operation

# Non-functional requirements implementation
(as per Sommerville)

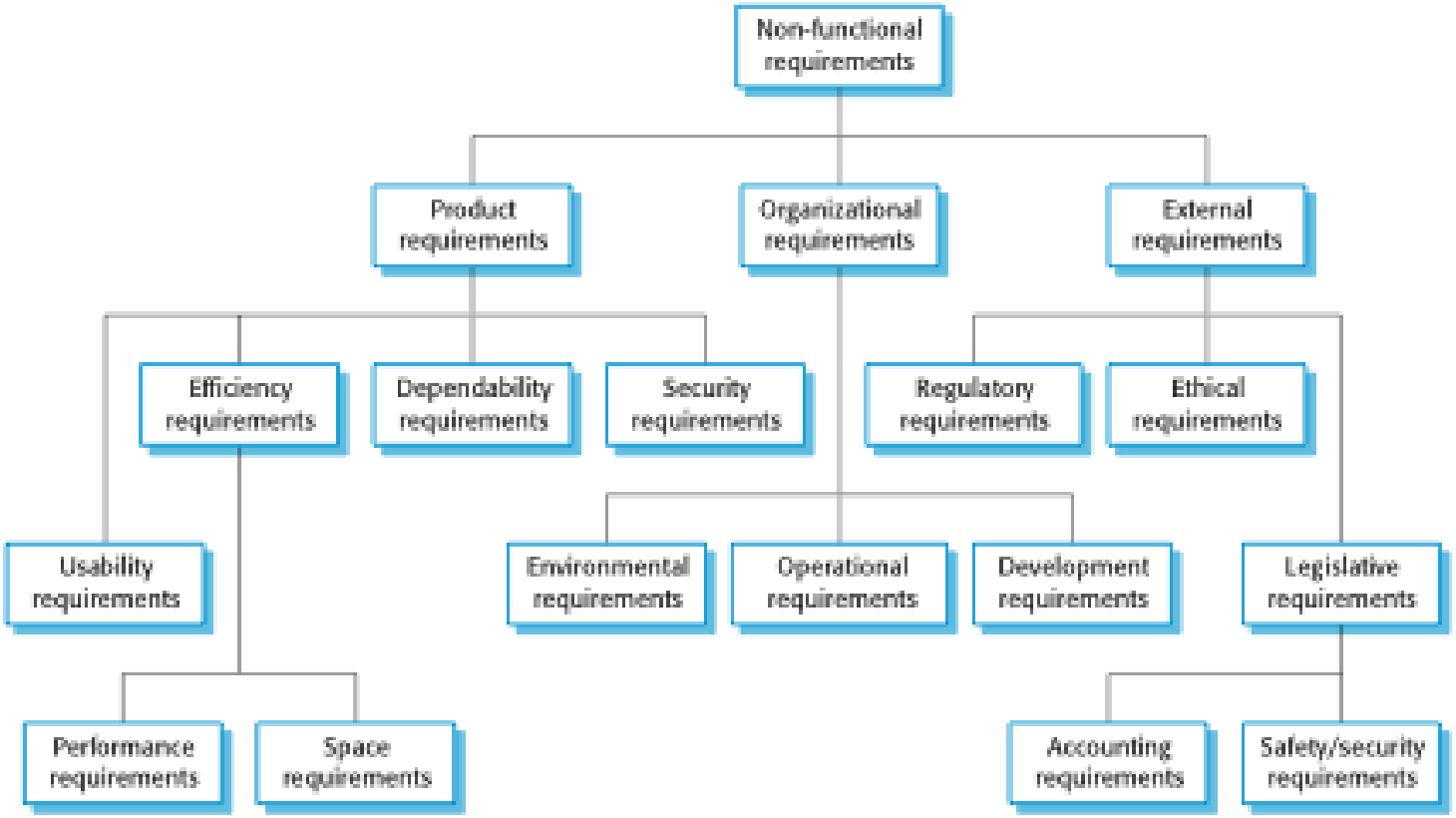
- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.

- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements.

# Non-functional classifications
(as per Sommerville)

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

# Types of nonfunctional requirement
(as per Sommerville)




Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

# Examples of nonfunctional requirements in PMS
(as per Sommerville)

**Product requirement**
The PMS(Patient Management System) shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

**Organizational requirement**
Users of the PMS system shall authenticate themselves using their health authority identity card.

**External requirement**
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

# Goals and requirements
(as per Sommerville)

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal
  - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.

# Usability requirements
(as per Sommerville)

- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)

- Medical staff shall be able to use all the system functions in PMS after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

# Metrics for specifying nonfunctional requirements
(as per Sommerville)

| Property | Measure |
|---|---|
| Speed | Processed transactions/second<br>User/event response time<br>Screen refresh time |
| Size | Mbytes<br>Number of ROM chips |
| Ease of use | Training time<br>Number of help frames |
| Reliability | Mean time to failure<br>Probability of unavailability<br>Rate of failure occurrence<br>Availability |
| Robustness | Time to restart after failure<br>Percentage of events causing failure<br>Probability of data corruption on failure |
| Portability | Percentage of target dependent statements<br>Number of target systems |

# Domain requirements

(as per Sommerville)

- The system's operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

# Train protection system
(as per Sommerville)

- This is a domain requirement for a train protection system:
- The deceleration of the train shall be computed as:
  - Dtrain = Dcontrol + Dgradient
  - where Dgradient is 9.81ms2 * compensated gradient/alpha and where the values of 9.81ms2 /alpha are known for different types of train.
- It is difficult for a non-specialist to understand the implications of this and how it interacts with other requirements.

# Domain requirements problems
(as per Sommerville)

- Understandability
  - Requirements are expressed in the language of the application domain;
  - This is often not understood by software engineers developing the system.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

# Requirements Modeling for WebApps

Content Analysis. The full spectrum of content to be provided by the WebApp is identified, including text, graphics and images, video, and audio data. Data modeling can be used to identify and describe each of the data objects.

Interaction Analysis. The manner in which the user interacts with the WebApp is described in detail. Use-cases can be developed to provide detailed descriptions of this interaction.

Functional Analysis. The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.

Navigation Model. Defines the overall navigation strategy for the WebApp

Configuration Analysis. The environment and infrastructure in which the WebApp resides are described in detail.

# When Do We Perform Analysis?

In some Web situations, analysis and design merge. However, an explicit analysis activity occurs when …

- the WebApp to be built is large and/or complex
- the number of stakeholders is large
- the number of Web engineers and other contributors is large
- the goals and objectives (determined during formulation) for the WebApp will effect the business' bottom line
- the success of the WebApp will have a strong bearing on the success of the business

# The Content Model

Content objects are extracted from use-cases

- examine the scenario description for direct and indirect references to content

Attributes of each content object are identified

The relationships among content objects and/or the hierarchy of content maintained by a WebApp

- Relationships—entity-relationship diagram or UML
- Hierarchy—data tree or UML

# The Interaction Model

- Composed of four elements:
  - use-cases
  - sequence diagrams
  - state diagrams
  - a user interface prototype

# The Functional Model

The functional model addresses two processing elements of the WebApp

- user observable functionality that is delivered by the WebApp to end-users
- the operations contained within analysis classes that implement behaviors associated with the class.

An activity diagram can be used to represent processing flow

# The Configuration Model

Server-side

- Server hardware and operating system environment must be specified
- Interoperability considerations on the server-side must be considered
- Appropriate interfaces, communication protocols and related collaborative information must be specified

Client-side

- Browser configuration issues must be identified
- Testing requirements should be defined

# Navigation Modeling-I

- Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be emphasized to force users to navigate in their direction?
- How should navigation errors be handled?
- Should navigation to related groups of elements be given priority over navigation to a specific element.
- Should navigation be accomplished via links, via search based access, or by some other means?
- Should certain elements be presented to users based on the context of previous navigation actions?
- Should a navigation log be maintained for users?

# Navigation Modeling-II

- Should a full navigation map or menu (as opposed to a single “back” link or directed pointer) be available at every point in a user’s interaction?
- Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebApp elements?
- Can a user “store” his previous navigation through the WebApp to expedite future usage?
- For which user category should optimal navigation be designed?
- How should links external to the WebApp be handled? overlaying the existing browser window? as a new browser window? as a separate frame?

# State of the Art for Large Applications

As per Capers Jones (of Software Productivity Research) best requirements practices include

- JAD (Joint Application Design)
- QFD (Quality Function Deployment)
- Security analysis & Vulnerability prevention
- Legacy Application Mining
- Joint Client/Vendor change control board
- Prototypes

# Why Mining Important

- As of 2009, more than half of application development is legacy replacement
- Usually legacy application specifications are out of date
- Legacy application business rules and algorithms need to be captured
- Data Mining of legacy code for hidden business rules and algorithms has to be part of requirements analysis

# Summary

- Requirements engineering helps software engineers better understand the problems they are trying to solve.
- Building an elegant computer solution that ignores the customer's needs helps no one.
- It is very important to understand the customer's wants and needs before you begin designing or building a computer-based solution.
- The requirements engineering process begins with inception, moves on to elicitation, negotiation, problem specification, and ends with review or validation of the specification.
- The intent of requirements engineering is to produce a written understanding of the customer's problem.
- Several different work products might be used to communicate this understanding (user scenarios, function and feature lists, analysis models, or specifications).