Sommerville Chapter 6, 7

Pressman Chapter 7

IEEE Standard 830

Objectives

- To explain the role of requirements in software engineering
- To describe the main activities in requirements engineering process
- To explain the challenges/risks in requirement engineering
- To describe the desirable characteristics of requirement specification

Topics covered

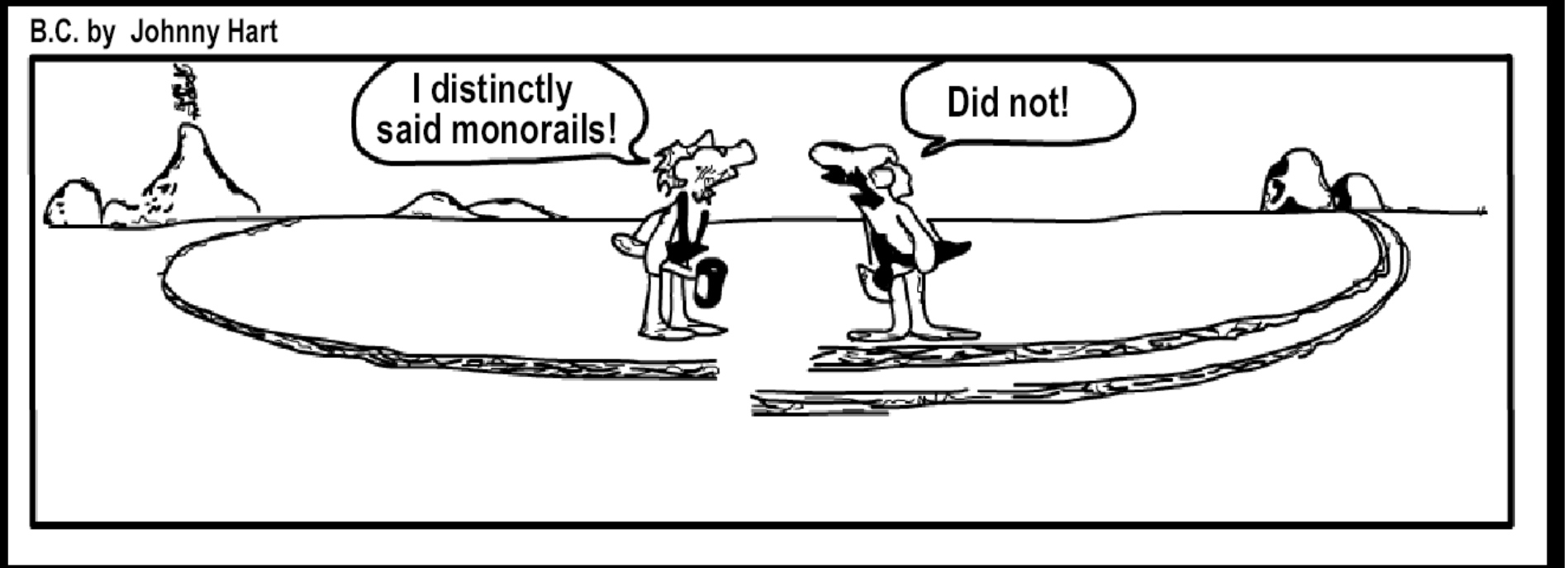
- Importance of requirements
- Requirements Engineering process
- Requirement Risks and Challenges
- Software Requirement Specification (SRS) Characteristics
- Requirement categorization

Announcement

- From today onwards, each student is required to give a 5-minutes presentation on an assigned topic.
- Marks will be given for
  - Relevance
  - Presentation Skills
  - Systematic approach

Why Requirements are Important

"I don't have time for requirements"



……But somehow there is always time to fix it later

Role of Requirements: A Contract

- The set of requirements constitutes a **contract** between the client and the software developer
- Serves as a basis for project planning (schedule, budget)
- Requirements document is used both to drive the design stage, and as a basis for test planning.

High Cost of Requirements Errors

If a unit cost of one is assigned to the effort required to detect and repair an error during the coding stage ….

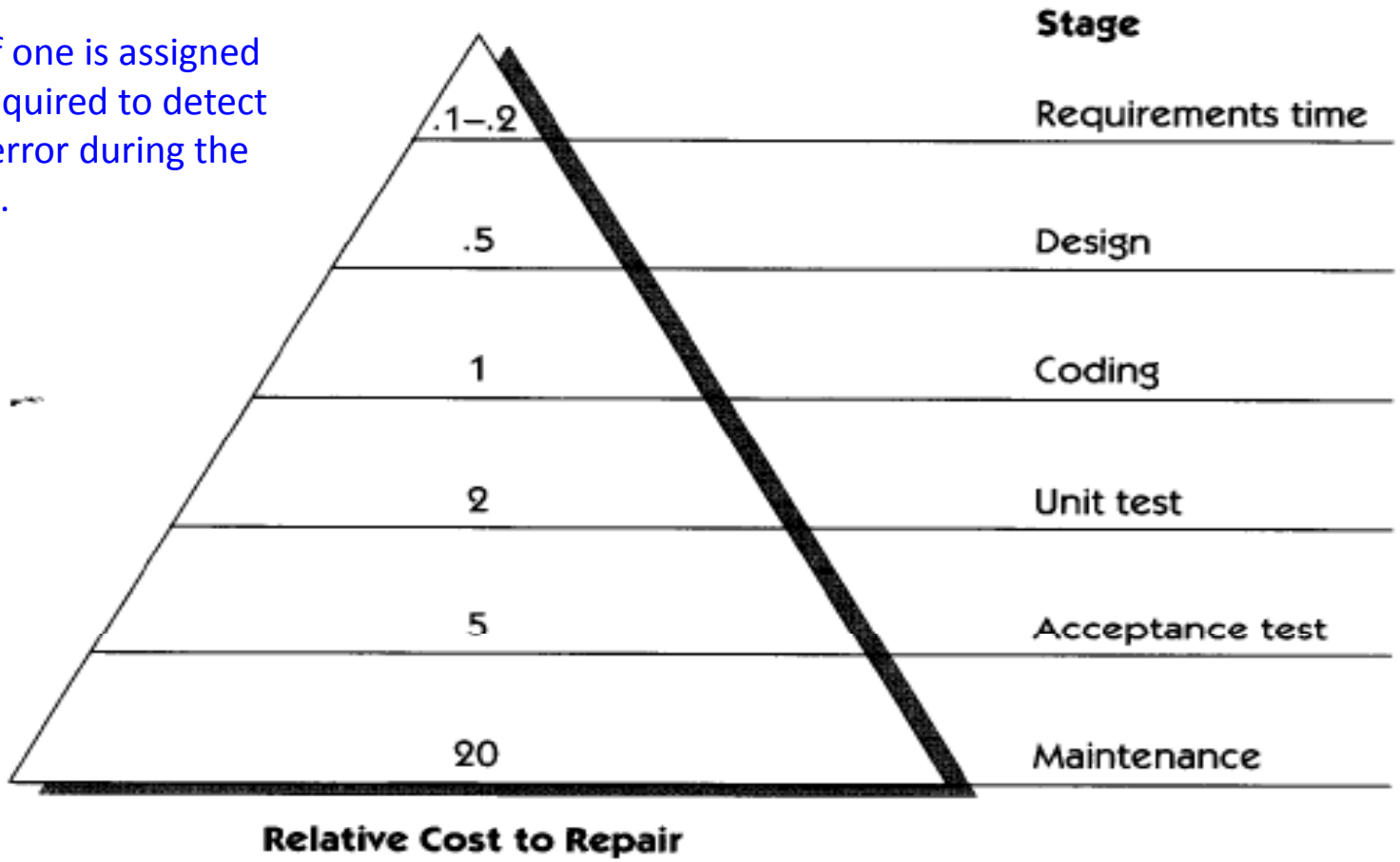


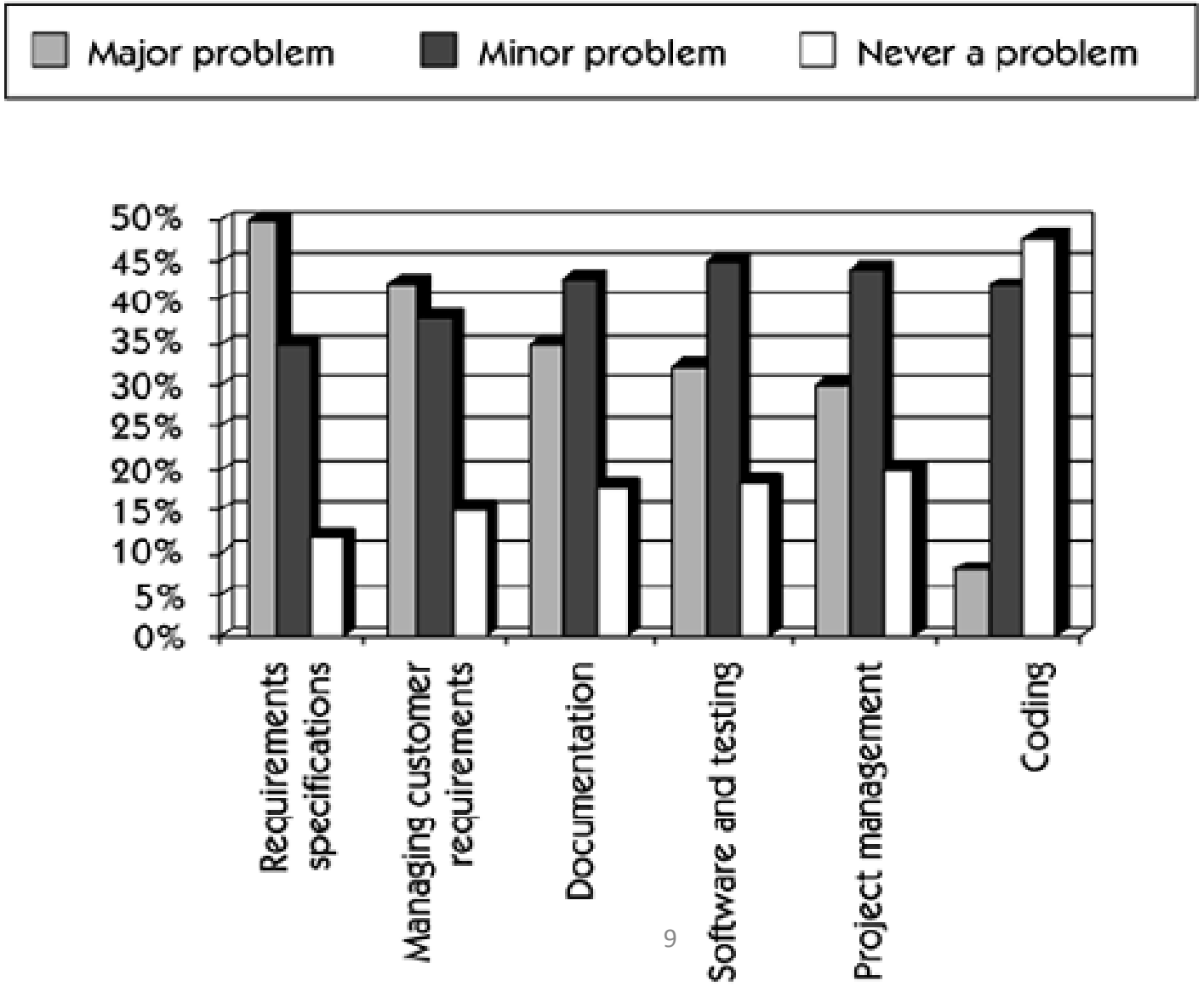
Figure 1–2 Relative cost to repair a defect at different lifecycle phases. (Data derived from Davis [1993].)

The Root Causes of Project Failure/Success (Standish)

- **The Three Largest Problems in Software Industry:**
  - Lack of user input: 13 percent of all "challenged" projects
  - Incomplete requirements and specifications: 12 percent of all "challenged" projects
  - Changing requirements : 12 percent of all "challenged" projects

- **The Three Primary Success Factors**:
  - User involvement: 16 percent of all successful projects
  - Executive management support: 14 percent of all successful projects
  - Clear statement of requirements: 12 percent of all successful projects

Largest Software Development Problems By Category



Requirement: Definition

- IEEE Standard Glossary of Software Engineering Terminology (1990) :
  1. A condition or capability needed by a user to solve a problem or achieve an objective.
  2. A condition or capability that must be met or possessed by a system component to satisfy a contract, standard, specification, or other formally imposed document.
  3. A documented representation of a condition or capability as in 1 or 2.

Requirement: Definition

- Sommerville and Sawyer, 1997:

  Requirements are…a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.

Requirements Engineering: Definition

- Zave, 1997:

  "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

Requirements Engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

Importance of RE

"The hardest single part of building a software system is **deciding what to build**. No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." -- Frederick P. Brooks

"Delivery is not necessarily the best time to discover the user requirements." (Urban Wisdom)

Importance of RE

"Done well, requirements engineering presents an opportunity to reduce costs and increase the quality of software systems. Done poorly, it could lead to a software project failure." –

Software Engineering Institute (SEI)

By Proper Requirements Engineering We Can…

- … know what the system is supposed to do
- … keep track of the current status of requirements
- … determine the impact of a requirements change

Requirement Engineering Steps

1. Inception
2. Elicitation
3. Analysis/Elaboration
4. Negotiations
5. Specification
6. Validation
7. Management

Inception

- Start of project
  - Business need
  - Market discovery
- Stakeholders from business community define a business case for the idea
  - Market
  - Feasibility
  - scope

Inception

- Discussions with a software engineering team start
- Software engineers use context free questions
  - The customer/end-user
  - The nature of desired solution
  - The effectiveness of preliminary communication

Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile.
- A short focused study that checks
  - If the system contributes to organisational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.

Feasibility study implementation

- Based on information assessment (what is required), information collection and report writing.
- Questions for carrying out the feasibility
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help business and requirements?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What must be supported by the proposed system and what not?

Context-Free Questions

- help us gain an understanding of the real problem without biasing the user's input.
- questions about the nature of the user's problem without context for a potential solution.
- Can be asked regardless of the nature of the project.
- These questions force us to listen before attempting to invent or describe a potential solution.

Context-Free Questions

- Listening gives us a better understanding of the customer's problem and any problems behind the problem.
- E.g.
  - What problems does this product solve?
  - What problems could this product create?
  - What environment is this product likely to encounter?

Requirements Elicitation and Analysis

- The process of identifying the needs and constraints of the various stakeholders for a software system
- Requirements elicitation is a process in which requirements are gathered for the new system to be developed

Elicitation Techniques

- Interviews
- Questionnaires
- Requirements workshops
- Storyboards
- Ethno-methodology
- Scenarios
- Use-cases

INTERVIEWING

- simple and direct technique that can be used in most circumstances.
- Convergence on some common needs will initiate a "requirements repository" for use during the project.
- Types:
  - Closed : pre-set agenda
  - Open-ended: no pre-set agenda
  - Normally a mix of closed and open-ended interviewing.

Interviews in practice

- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

Interviews

- Make sure that the biases and predispositions of the interviewer do not interfere with a free exchange of information.
- We must not let our context (background) interfere with understanding the real problem to be solved

Questionnaires

- There is no substitute for an interview.
- Although the questionnaire technique is often used and appears scientific because of the opportunity for statistical analysis of the quantitative results, the technique is not a substitute for interviewing.
- When it comes to requirements gathering, the questionnaire technique has some fundamental problems.
  - E.g. Relevant questions cannot be decided in advance.

Questionnaires

- It is difficult to explore new domains ("What you really should be asking about is . . ."), and there is no interaction to explore domains that need to be explored.
- It is difficult to follow up on unclear user responses.
- Indeed, some have concluded that the questionnaire technique suppresses almost everything good about requirements gathering.

Questionnaires

- However, the questionnaire technique can be applied with good effect as a corroborating technique after the initial interviewing and analysis activity.
- For example, if the application has a large number of existing or potential users and if the goal is to provide statistical input about user or customer preferences among a limited set of choices, a questionnaire can be used effectively to gather a significant amount of focused data in a short period of time.
- In short, the questionnaire technique, like all elicitation techniques, is suited to a subset of the requirements challenges that an organization may face.

Requirements Workshops

- The requirements workshop may be the most powerful technique for eliciting requirements.
- If we were to be given only one requirements elicitation technique—one that we had to apply in every circumstance, no matter the project context, no matter what the time frame—we would pick the requirements workshop.
- The requirements workshop is designed to encourage consensus on the requirements of the application and to gain rapid agreement on a course of action, all in a very short time.

Requirements Workshops

- It gathers all key stakeholders together for a short but intensely focused period (1-2 days)
- The use of an outside facilitator experienced in requirements elicitation can help ensure the success of the workshop.
- Brainstorming is the most important part of the workshop.

Requirements Workshops

- It can expose and resolve political issues that are interfering with project success.
- The output, a preliminary system definition at the features level, is available immediately.
- The workshop is facilitated by a team member or, better yet, by an experienced outside facilitator and focuses on the creation or review of the high-level features to be delivered by the new application.

Sample Agenda for Requirements Workshop

Time	Agenda Item	Description
8:00–8:30	Introduction	Review agenda, facilities, and rules
8:30–10:00	Context	Present project status, market needs, results of user interviews, and so on
10:00–12:00	Brainstorming	Brainstorm features of the application
12:00–1:00	Lunch	Work through lunch to avoid loss of momentum
1:00–2:00	Brainstorming	Continue brainstorming
2:00–3:00	Feature definition	Write out two- or three-sentence definitions for features
3:00–4:00	Idea reduction and prioritization	Prioritize features
4:00–5:00	Wrap-up	Summarize and assign action items

Storyboarding

- The purpose of storyboarding is to elicit early "Yes, But" reactions.
- Storyboards identify the players, explain what happens to them, and describe how it happens.
- Make the storyboard sketchy, easy to modify, and not shippable.
- Storyboard early and often on each project with new or innovative content.

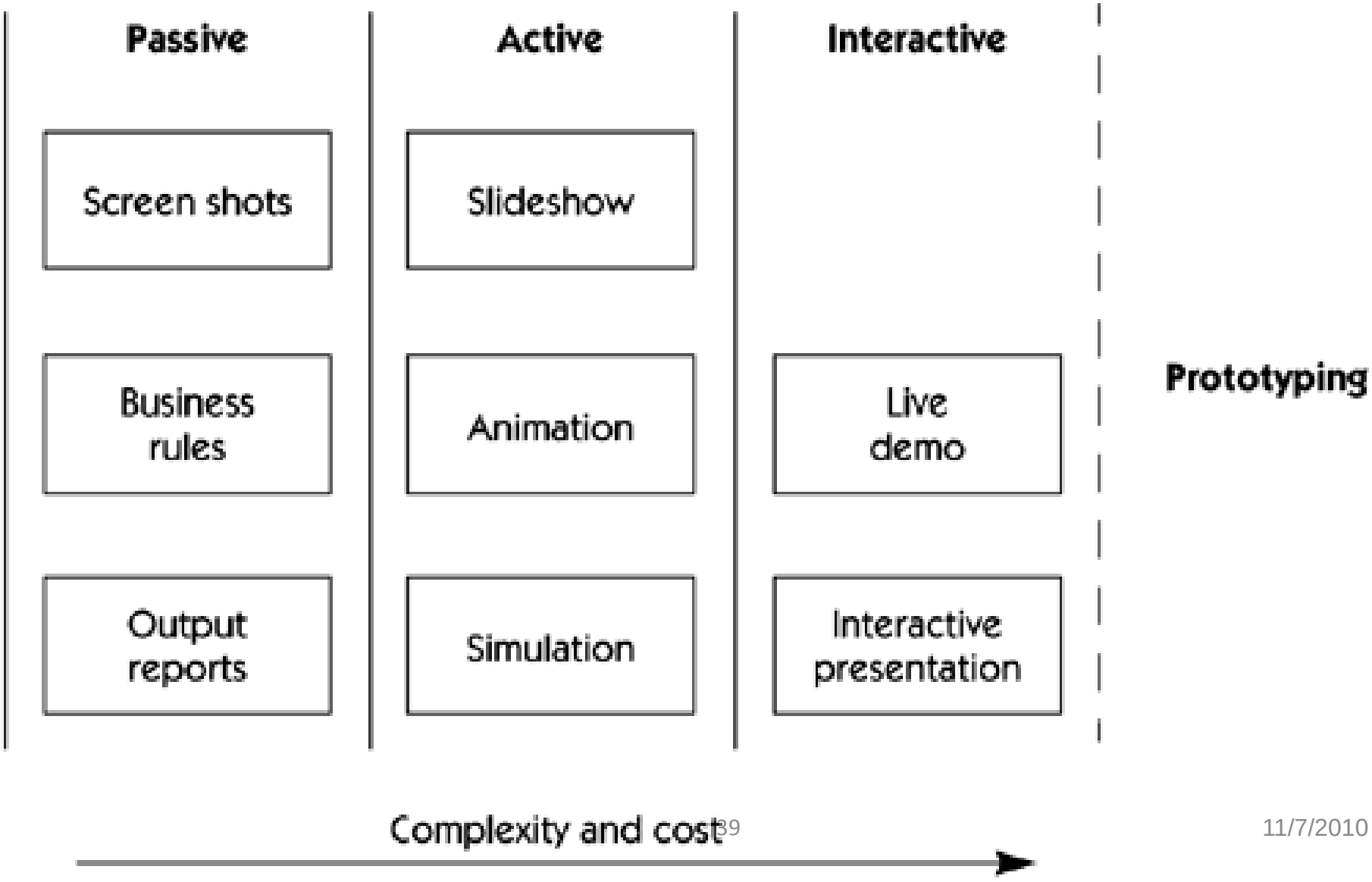
Storyboarding

- Is extremely inexpensive
- Is user friendly, informal, and interactive
- Provides an early review of the user interfaces of the system
- Is easy to create and easy to modify

Storyboarding

- Storyboards also offer an excellent way to ease the "blank-page" syndrome. When the users do not know what they want or have trouble envisioning any solution to the current problem, even a poor storyboard is likely to elicit a response of "No, that's not what we meant, it's more like the following," and the game is on.

Types of Storyboards



Tips for Storyboarding

- Don't invest too much in a storyboard. Customers will be intimidated about making changes if it looks like a real work product or they think they might insult you, a particularly difficult problem in some cultures.
- If you don't change anything, you don't learn anything. Make the storyboard easy to modify. You should be able to modify a storyboard in a few hours.

Tips for Storyboarding

- Don't make the storyboard too functional. If you do, some stakeholders may want you to "ship it.“
- Whenever possible, make the storyboard interactive. The customer's experience of use will generate more feedback and will elicit more new requirements than a passive storyboard will.

Ethno-methodology

- Entails the analyst spending a long period of time with the organization and making detailed observations about its work practices.
- Subsequent analysis of the observations can reveal vital information about the organization, which usually differs markedly from the one recorded in formal documents (manuals, handbooks) of the organization.
- The advantage of the ethnography approach over conventional systems analysis lies on the fact that analysts are passive observers and do not try to impose their judgments on the practices which are observed.

Scenarios

- Scenarios are real-life examples of how a system can be used.
- An integral part of agile methods
- They should include
  - A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.

Example: LIBSYS scenario for article downloading (1)

Initial assumption: The user has logged on to the LIBSYS system and has located the journal containing the copy of the article.

Normal: The user selects the article to be copied. He or she is then prompted by the system to either provide subscriber information for the journal or to indicate how they will pay for the article. Alternative payment methods are by credit card or by quoting an organisational account number.

The user is then asked to fill in a copyright form that maintains details of the transaction and they then submit this to the LIBSYS system.

The copyright form is checked and, if OK, the PDF version of the article is downloaded to the LIBSYS working area on the userÕscomputer and the user is informed that it is available. The user is asked to select a printer and a copy of the article is printed. If the article has been flagged as Õprint-onlyÕit is deleted from the userÕs system once the user has confirmed that printing is complete.

LIBSYS scenario for article downloading (2)

What can go wrong: The user may fail to fill in the copyright form correctly. In this case, the form should be re-presented to the user for correction. If the resubmitted form is still incorrect then the userÕsrequest for the article is rejected.

The payment may be rejected by the system. The userÕs request for the article is rejected.

The article download may fail. Retry until successful or the user terminates the session.

It may not be possible to print the article. If the article is not flagged as Ōprint-onlyÕthen it is held in the LIBSYS workspace. Otherwise, the article is deleted and the userÕs account credited with the cost of the article.

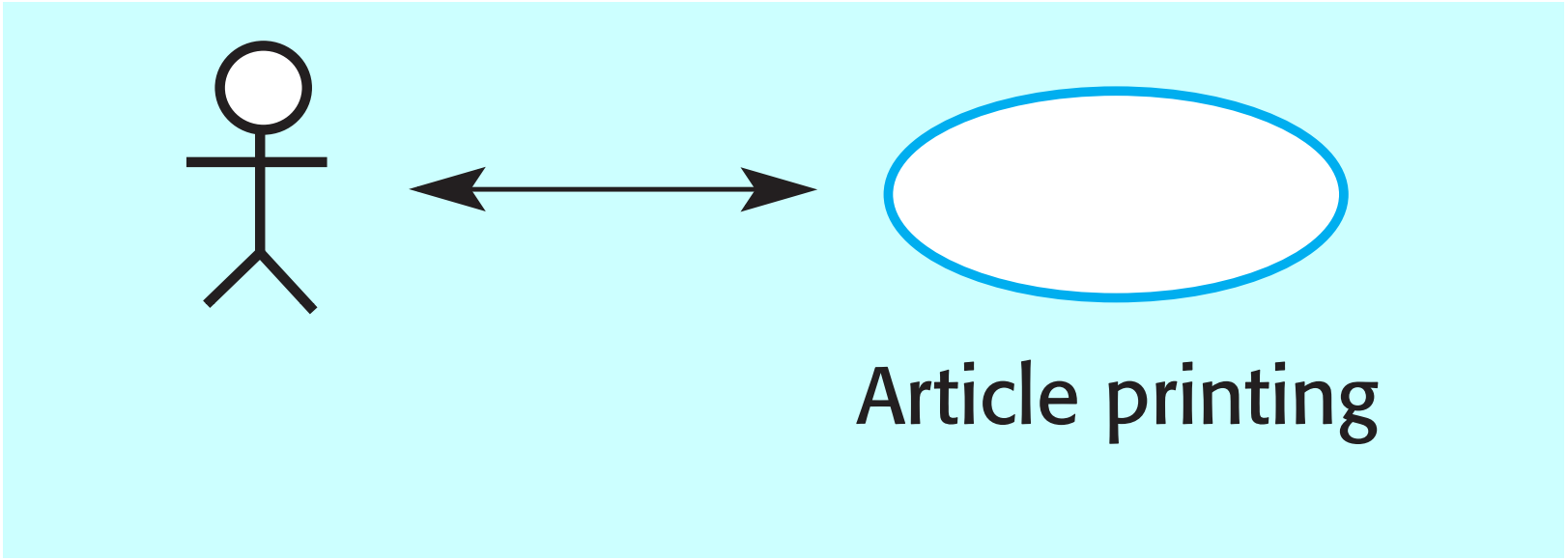
Other activities: Simultaneous downloads of other articles.

System state on completion: User is logged on. The downloaded article has been deleted from LIBSYS workspace if it has been flagged as print-only.

Use cases

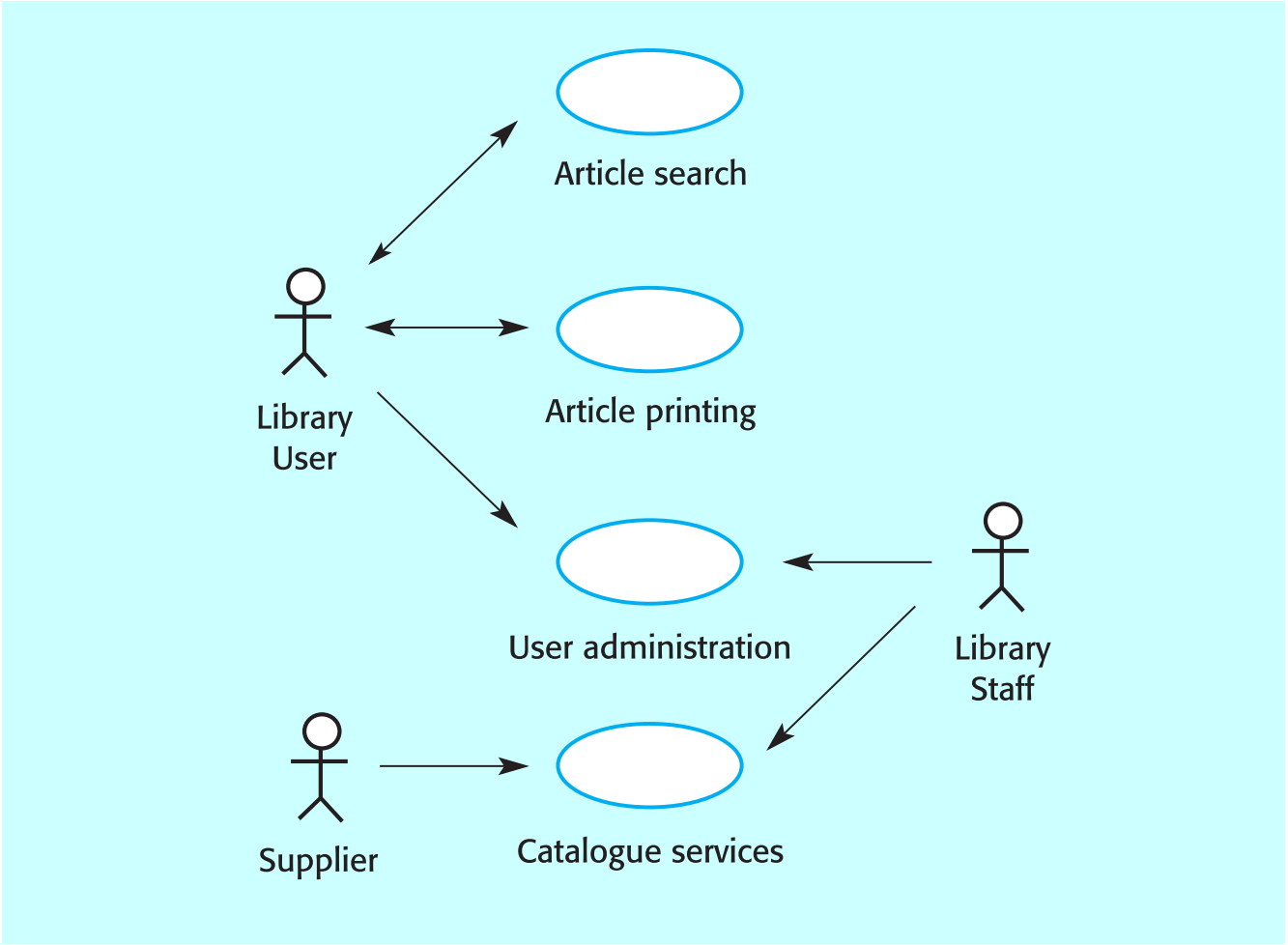
- Use-cases are a scenario based technique in Unified Modeling Language (UML) which identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

Article printing use-case (Top Level)

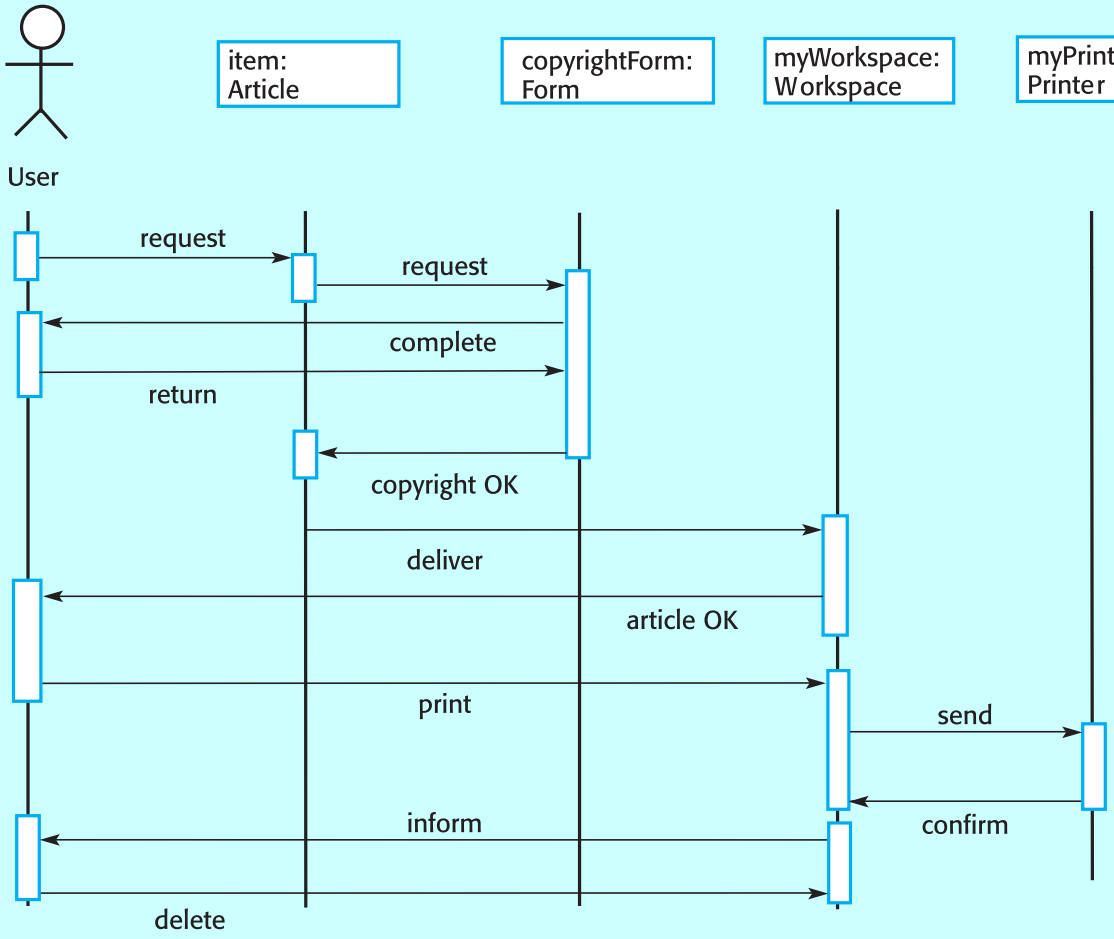


stick figure: Actor
Ellipse: a distinct class/category of interaction

Complete LIBSYS system: all use cases



sequence diagram: Print article



Barriers to Elicitation

- The "User and the Developer" Syndrome
  - arises from the communication gap between the user and the developer.
  - Users and developers are typically from different worlds, may even speak different languages, and have different backgrounds, motivations, and objectives
  - Ease of omitting “obvious” information by users

Barriers to Elicitation

- Users have incomplete understanding of their needs
- Users have poor understanding of computer capabilities and limitations
- Analysts have poor knowledge of problem domain

Negotiation

- Customer may be asked to balance functionality, performance, and other product characteristics against cost and time to market.
- The intent is to develop a project plan that meets the need of the customer while at the same time reflecting the real-world constraints (e.g. time, people, budget etc.)
- The best negotiations strive for win-win results

Specification

- Document the requirements and associated work products.
- Several desirable characteristics of good specification.

Characteristics of INDIVIDUAL Requirement Statements

- Complete
- Correct
- Feasible
- Necessary
- Prioritized
- Unambiguous
- Verifiable

Complete

- Each requirement must fully describe the functionality to be delivered.
- It must contain all the information necessary for the developer to design and implement that bit of functionality.
- If you know you're lacking certain information, use *TBD* (to be determined) as a standard flag to highlight these gaps. Resolve all TBDs in each portion of the requirements before you proceed with construction of that portion.

Correct

- Each requirement must accurately describe the functionality to be delivered.
- The reference for correctness is the source of the requirement, such as an actual customer or a higher level system requirements specification.
- Only user representatives can determine the correctness of user requirements.

Feasible

- It must be possible to implement each requirement within the known capabilities and limitations of the system and its environment.
- To avoid infeasible requirements, have a developer work with the requirements analysts or marketing personnel throughout the elicitation process.
- Incremental development approaches and proof-of-concept prototypes are ways to evaluate requirement feasibility.

Necessary

- Each requirement originates from a source you recognize as having the authority to specify requirements.
- Trace each requirement back to its origin, such as a use case, system requirement, regulation, or some other voice-of-the-customer input.

PRIORITIZED

- Assign an implementation priority to each functional requirement, feature, or use case to indicate how essential it is to a particular product release.
- If all the requirements are considered equally important, it's hard for the project manager to respond to budget cuts, schedule overruns, personnel losses, or new requirements added during development.

Unambiguous

- The reader of a requirement statement should be able to draw only one interpretation of it.
  - Natural language is highly prone to ambiguity
  - Effective ways to reveal ambiguity include:
    - **formal inspections** of the requirements specifications,
    - writing **test cases from requirements**
    - creating **user scenarios** that illustrate the expected behavior of a specific portion of the product.

Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘appropriate viewers for each document’
  - User intention - special purpose viewer for each different document type;
  - Developer interpretation - Provide a text viewer that shows the contents of the document.

Verifiable

- See whether you can devise tests or use other verification approaches, such as inspection or formal specification, to determine whether each requirement is properly implemented in the product.
- Requirements that are not consistent, feasible, or unambiguous also are not verifiable.

Example: VERIFIABLE

- *An operator shall not have to wait for the transaction to complete… change to measureable*
- *An operator shall not have to wait more than 1s for a transaction to complete for at least 95% of the transactions.*

Interesting phenomenon: What gets measured gets done.

Requirements measures

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

Characteristics of Requirement Specification SET (SRS)

- Complete
- Consistent
- Modifiable
- Traceable

COMPLETE SRS

- No requirements or necessary information should be absent.
- Missing requirements are hard to spot because they aren't there!
- Focusing on user tasks, rather than on system functions, can help you to prevent incompleteness.

Consistent SRS

- Consistent requirements do not conflict with other software requirements or with higher level (system or business) requirements.
- Inconsistencies can slip in undetected if you review only the specific change and not any related requirements.
- Recording the originator of each requirement lets you know who to talk to if you discover conflicts.

Requirements interaction

- Conflicts between different non-functional requirements are common in complex systems.
- Example: Spacecraft system
  - To minimise weight, the number of separate chips in the system should be minimised.
  - To minimise power consumption, lower power chips should be used.
  - However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

Modifiable SRS

- You must be able to revise the SRS when necessary and to maintain a history of changes made to each requirement.
- Each requirement be uniquely labeled and expressed separately from other requirements so you can refer to it unambiguously.
- Organizing requirements so that related requirements are grouped together.
- Each requirement should appear only once in the SRS.
- Create a table of contents, index, cross-reference listing, traceability matrices

Traceable SRS

- A traceable requirement can be linked backward to its origin and forward to the design elements and source code that implement it and to the test cases that verify the implementation as correct.
- Traceable requirements are uniquely labeled and are written in a structured, fine-grained way, as opposed to large, narrative paragraphs or bullet lists.
- Avoid lumping multiple requirements together into a single statement; the different requirements might trace to different design and code elements.

Exercise 1

"*The product shall provide status messages at regular intervals not less than every 60 seconds.*"

Quality Analysis?

Exercise 1: Improved

1. The Background Task Manager shall display status messages in a designated area of the user terface at intervals of 60 plus or minus 10 seconds.

2. If background task processing is progressing normally, the percentage of the background task rocessing that has been completed shall be displayed.

3. A message shall be displayed when the background task is completed.

4. An error message shall be displayed if the background task has stalled."

Exercise 2

"The product shall switch between displaying and hiding non-printing characters instantaneously."

Analysis?

Exercise 2: Improved

"The user shall be able to toggle between displaying and hiding all HTML markup tags in the document being edited with the activation of a specific triggering condition."

Exercise 3

"Charge numbers should be validated on-line against the master corporate charge number list, if possible."

Analysis?

Exercise 3: Improved

"The system shall validate the charge number entered against the online master corporate charge number list.

If the charge number is not found on the list, an error message shall be displayed and the order shall not be accepted."

Problems with natural language specification of requirements

- Lack of clarity
  - Precision is difficult without making the document difficult to read.
- Requirements confusion
  - Different types of requirements tend to be mixed-up.
- Requirements amalgamation
  - Several different requirements may be expressed together as a single requirement.

Problems with NL specification of system requirements

- Ambiguity
  - The readers and writers of the requirement must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult.
- Over-flexibility
  - The same thing may be said in a number of different ways in the specification.
- Lack of modularisation
  - NL structures are inadequate to structure system requirements.

Guidelines for writing requirements

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

Alternatives to NL specification for system requirements

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. Use-case descriptions and sequence diagrams are commonly used examples.
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Structured language specifications

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.
- The advantage is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification.

Form-based specifications

- 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 post conditions (if appropriate).
- The side effects (if any) of the function.

Form-based node specification

Insulin Pump/Control Software/SRS/3.3.2	
Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose Š the dose in insulin to be delivered
Destination	Main control loop
Action:	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin..
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side-effects	None

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

Tabular specification

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2-r1)<(r1-r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing. ((r2-r1) □ (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

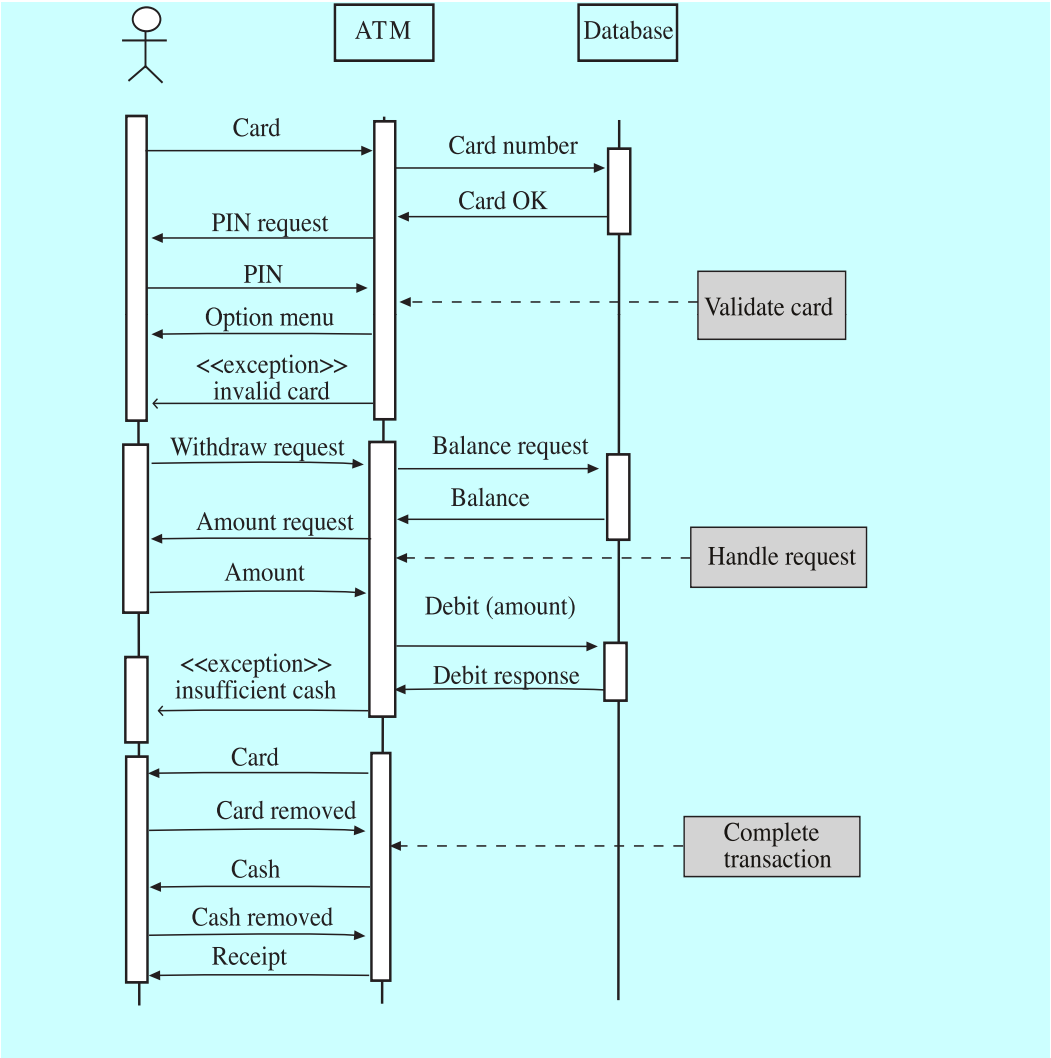
Graphical models

- Graphical models are most useful when you need to show how the system state changes or where you need to describe a sequence of actions.

Sequence diagrams

- These show the sequence of events that take place during some user interaction with a system.
- You read them from top to bottom to see the order of the actions that take place.
- Example: Cash withdrawal from an ATM
  - Validate card;
  - Handle request;
  - Complete transaction.

Sequence diagram of ATM withdrawal



The requirements document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should be a set of WHAT the system should do rather than HOW it should do it

IEEE 830

- Defines a generic structure for a requirements document that must be instantiated for each specific system.
  - Introduction.
    - Purpose of requirements document
    - Scope of the product
    - Definitions, acronyms, abbreviations
    - Overview of remainder of the document
  - General description.
    - Product perspective
    - Product functions
    - User characteristics
    - General constraints
    - Assumptions and dependencies

IEEE requirements standard guidelines (IEEE 830)

- Specific requirements.
  - Functional requirements
  - Non-functional requirements
  - Interface requirements
- Appendices.
- Index.

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements validation techniques

- 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.

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development/use.
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

Requirements change

- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

Requirements management planning

- During the requirements engineering process, you have to plan:
  - Requirements identification
    - How requirements are individually identified;
  - A change management process
    - The process followed when analysing a requirements change;
  - Traceability policies
    - The amount of information about requirements relationships that is maintained;
  - CASE tool support
    - The tool support required to help manage requirements change;

Traceability

- Traceability is concerned with the relationships between requirements, their sources and the system design
- Source traceability
  - Links from requirements to stakeholders who proposed these requirements;
- Requirements traceability
  - Links between dependent requirements;
- Design traceability
  - Links from the requirements to the design;

A traceability matrix

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

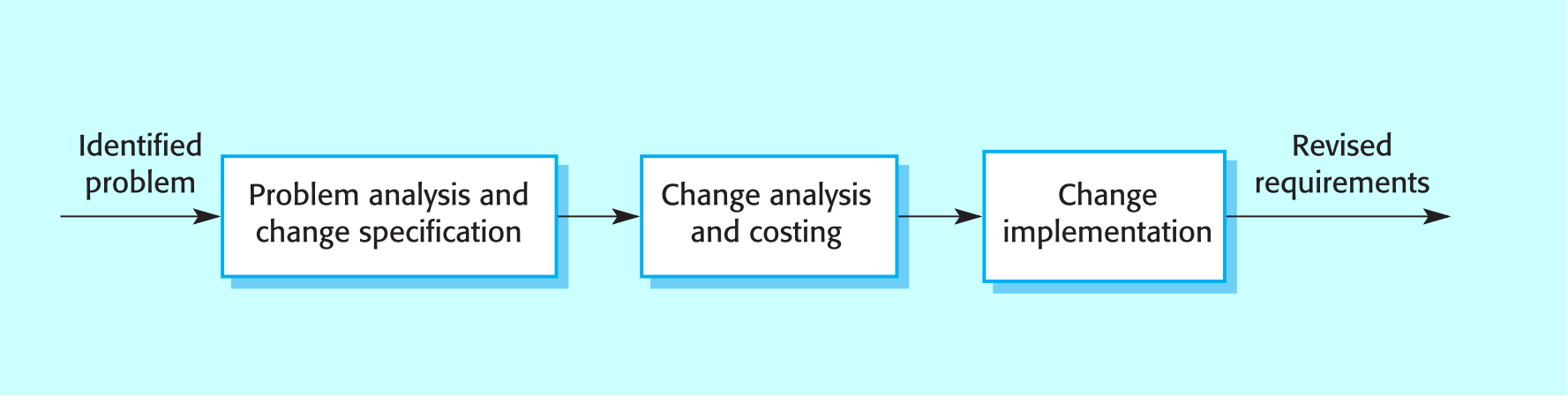
CASE tool support

- Requirements storage
  - Requirements should be managed in a secure, managed data store.
- Change management
  - The process of change management is a workflow process whose stages can be defined and information flow between these stages partially automated.
- Traceability management
  - Automated retrieval of the links between requirements.

Requirements change management

- Should apply to all proposed changes to the requirements.
- Principal stages
  - Problem analysis. Discuss requirements problem and propose change;
  - Change analysis and costing. Assess effects of change on other requirements;
  - Change implementation. Modify requirements document and other documents to reflect change.

Change management



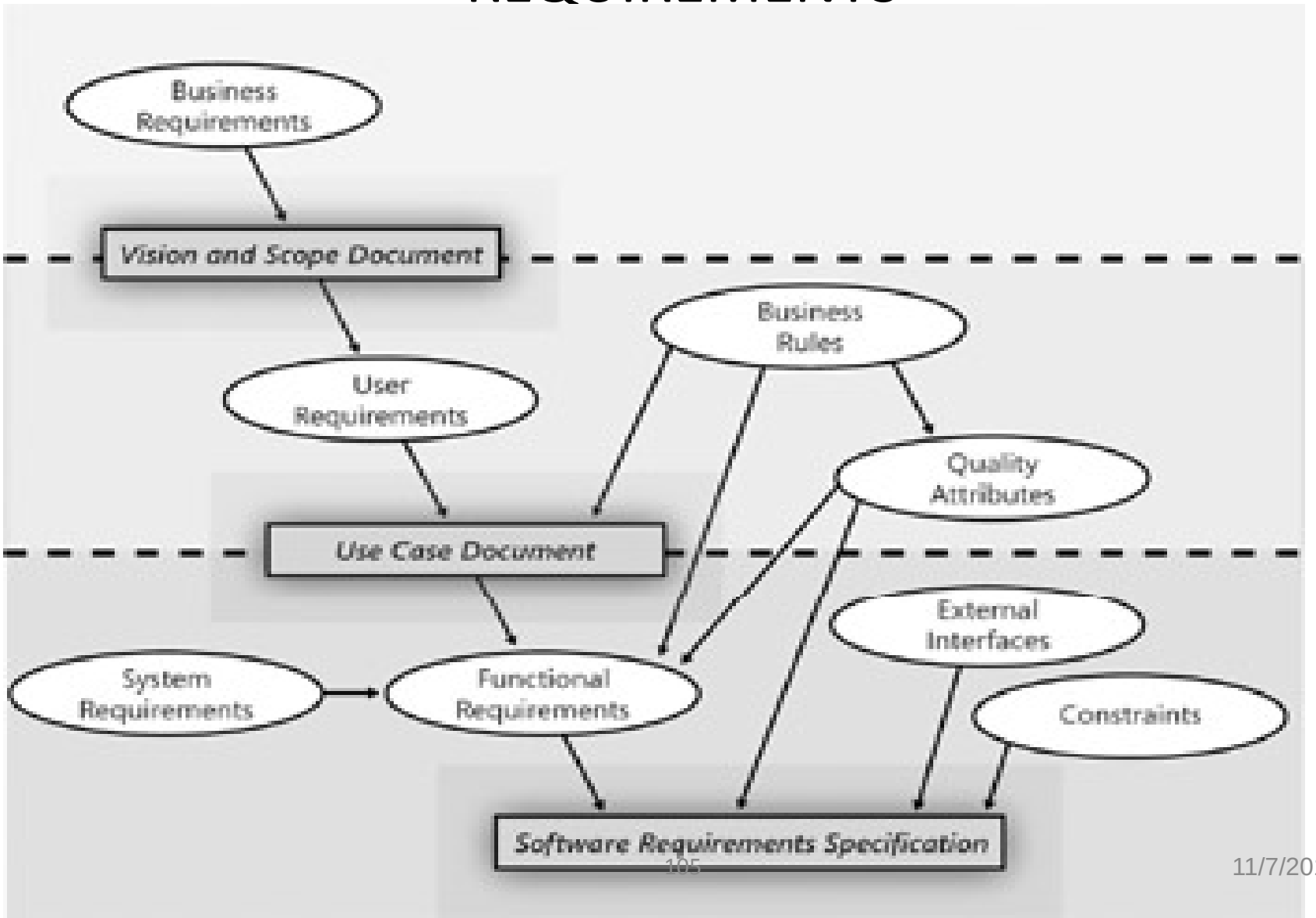
Requirements: Main Risks

- Insufficient user involvement
- Creeping user requirements
- Ambiguous requirements
- Gold plating
- Minimal specification
- Overlooked user classes

Benefits from a High-Quality Requirements Process

- Fewer requirement defects
- Reduced development rework
- Fewer unnecessary features
- Lower enhancement costs
- Faster development
- Fewer miscommunications
- Reduced scope creep
- Reduced project chaos
- More accurate system testing estimates
- Higher customer and team member satisfaction

LEVEL-BASED CATEGORIZATION OF REQUIREMENTS



BUSINESS REQUIREMENTS

- Represent high-level objectives of the organization or customer who requests the system.
- Typically come from the funding sponsor for a project, the acquiring customer, the manager of the actual users, the marketing department, or a product visionary.
- Describe why the organization is implementing the system—the objectives the organization hopes to achieve.
- Recorded in a *vision and scope document*, sometimes called a *project charter* or a *market requirements document*.
- Defining the project scope is the first step in controlling the common problem of scope creep.

USER REQUIREMENTS

- Describe user goals or tasks that the users must be able to perform with the product.
- Valuable ways to represent user requirements include use cases and scenario descriptions.
- An example of a use case is "Make a Reservation" using an airline, a rental car, or a hotel Web site.

User requirements

- Users don’t have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

FUNCTIONAL REQUIREMENTS

- Specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements.
- Functional requirements describe what the developer needs to implement.
- Almost any action, calculate, inspect, publish, or most other active verbs can be a functional requirement.
- Example: *The product shall predict which road sections will freeze within the selected time parameters.*

SYSTEM REQUIREMENTS

- The term *system requirements* describes the top-level requirements for a product that contains multiple subsystems—that is, a *system* (IEEE 1998c).
- A system can be all software or it can include both software and hardware subsystems. People are a part of a system, too, so certain system functions might be allocated to human beings.

BUSINESS RULES

- *Business rules* include corporate policies, government regulations, industry standards, accounting practices, and computational algorithms.
- business rules are not themselves software requirements because they exist outside the boundaries of any specific software system. However, they often restrict who can perform certain use cases or they dictate that the system must contain functionality to comply with the pertinent rules.
- Sometimes business rules are the origin of specific quality attributes that are implemented in functionality.

QUALITY ATTRIBUTES

- In addition to the functional requirements, the SRS contains nonfunctional requirements. These include performance goals and descriptions of quality attributes.
- *Quality attributes* augment the description of the product's functionality by describing the product's characteristics in various dimensions that are important either to users or to developers.
- These characteristics include usability, portability, efficiency, and robustness etc.

Examples: NFRs

- The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.

- The system shall provide access to the legacy course catalogue database with no more than a 10 second latency.

OTHER NON-FUNCTIONAL REQUIREMENTS

- External interfaces between the system and the outside world.
- *Constraints* that impose restrictions on the choices available to the developer for design and construction of the product.
- Constraints do not affect the external behavior of the system, but must be fulfilled to meet technical, business, or contractual obligations.

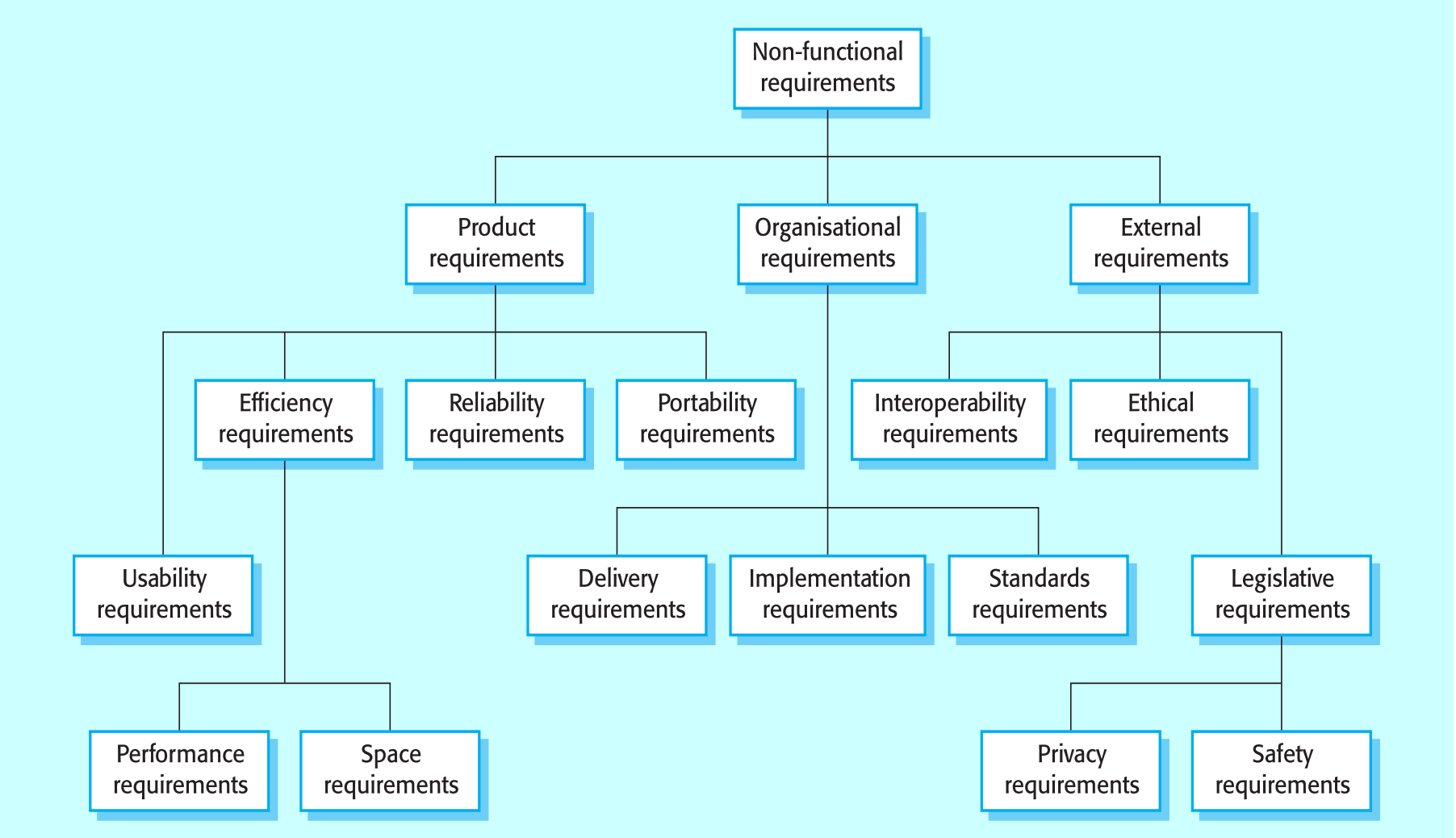
Example: Constraints

- Example:
  - The system shall integrate with existing legacy system (course catalog database) which operates on the College DEC VAX Main Frame.

Non-functional requirements classification

- 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.

Non-functional requirement types



Non-functional requirements examples

- Product requirement

  8.1 The user interface for LIBSYS shall be implemented as simple HTML without Java applets.

- Organisational requirement

  9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.

- External requirement

  7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

Domain Requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain.
- Domain requirements may be new functional requirements, constraints on existing requirements or may define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Example: Library system domain requirements

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.
- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Example 2: Train protection system domain requirements

- The deceleration of the train shall be computed as:
  - $D_{train} = D_{control} + D_{gradient}$

  where $D_{gradient}$ is $9.81ms^2$ * compensated gradient/alpha and where the values of $9.81ms^2$ /alpha are known for different types of train.

Domain requirements problems

- 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 and Design

- In principle, requirements should state what the system should do and the design should describe how it does this.
- In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements;
  - The system may inter-operate with other systems that generate design requirements;
  - The use of a specific architecture to satisfy non-functional requirements (such as N-version programming to achieve reliability)

Key points

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.

Key points

- System requirements are intended to communicate the functions that the system should provide.
- A software requirements document is an agreed statement of the system requirements.
- The IEEE standard 830 is a useful starting point for defining more detailed specific requirements standards.

Reading Assignment

- Read IEEE SRS standard (IEEE 830)