ECE 651 – Software Engineering
Week Two: Requirements
January 27th, 2016

Ric Telford
Adjunct Associate Professor

Founder, Telford Ventures

Duke | PRATT SCHOOL OF ENGINEERING

01/27/16 – Week 2 Overview

- Recap / Announcements
- Homework review / Java / GitLab
- Software Engineering Overview - Recap and Continuation (Lecture)
- Requirements (Lecture)
- Requirements (Exercise)
- About Week 3

Recap / Announcements

- We will go back over the Overview slides quickly today and then start into Requirements
- Time to start closing in on a Team and a Project. Team and initial Project decision will be due next week, **but you have opportunity to change Project**
  - If you have any questions about the Projects on the Project List, see me this week.
  - I am not opposed to having 2 teams on a larger project if the responsibility can be clearly split.
- We are down to 41 students, so if you would prefer a team of 2 people let me know ASAP.
- For coding homework assignments going forward, we will have a specific GitLab Group set up. For those of you that did your homework before we had "Homework1" you didn't have to move it to get graded but it would be nice now if you did.

Teams

Team Name	Members	Project
	Xiaodan Zhu, Wankun Zhu, Shuai Yuan	

Homework 1

- Great job everyone!
- A lot of inventive ideas, great reusable code
- I have summarized the projects with the owners userid in case you want to reuse some of this code in your project.
  - Reuse is fine for code that is already in public domain
  - Please give attribution as applicable
  - Summary will be in Resources under this week
- Any questions about the homework, GitLab, Java, etc?

Software Engineering

- Because of their abstract nature, Software Systems can be:
  - Complex
  - Difficult to understand
  - Expensive to change
- Due to the fact that almost everything requires software these days, there is no one process that covers all types of software.
- But, there is a general description of the processes that need to be followed and tailored for the specific project through all phases
- The *software lifecycle* is the progression of software through these phases
- *Software Engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance*.

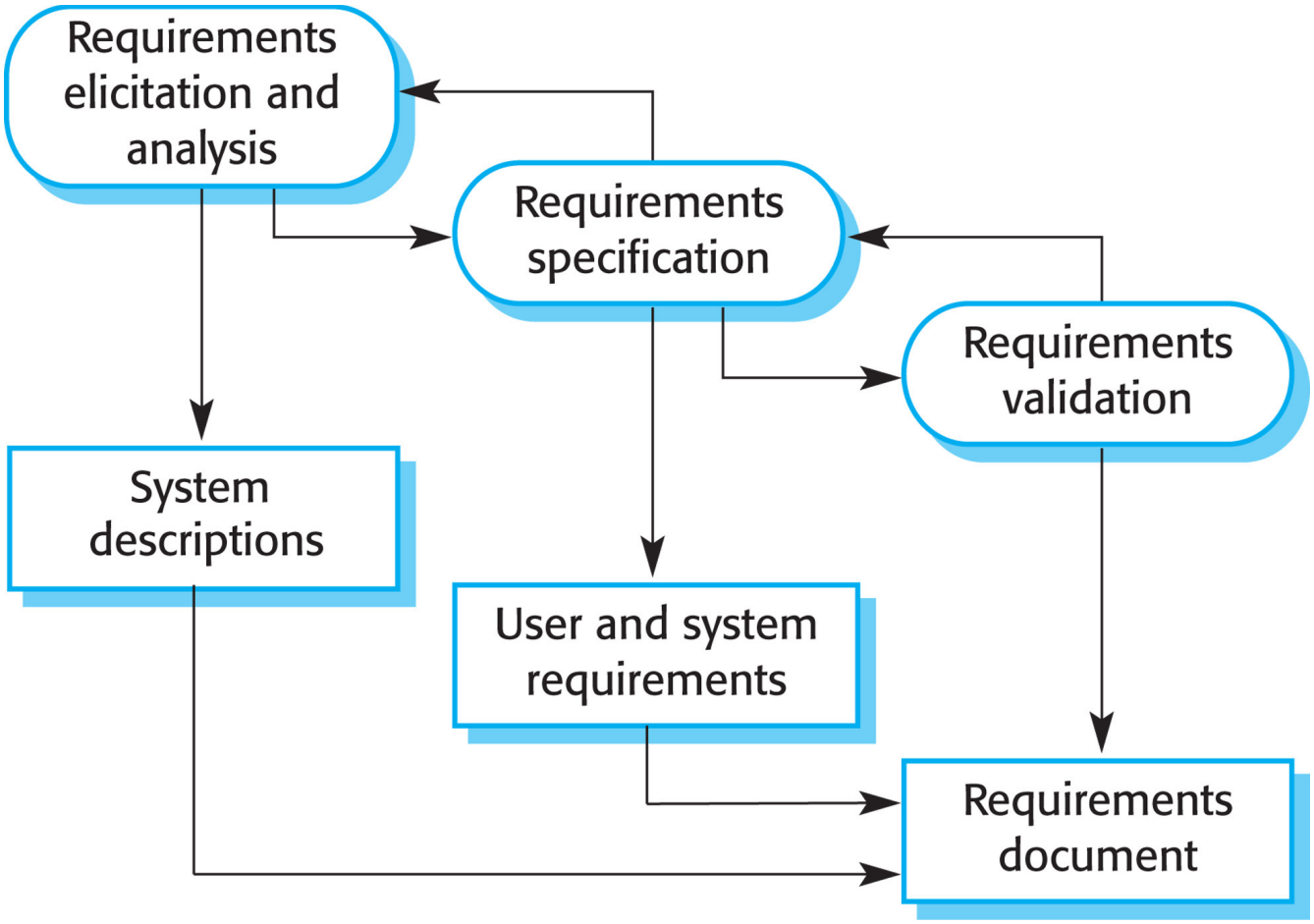
The Software Lifecycle

- There are *4 fundamental activities* in the software lifecycle:
  - Specification – definition of what is to be created and how deployed
  - Development – creation of the software – design and code
  - Validation – testing, quality control, verification that it meets rqmts
  - Evolution – maintenance, enhancements due to changing requirements
- Software engineering helps define processes for each activity
- In addition to having defined processes, any type of software (standalone, embedded, web, mobile, etc) needs to:
  - Have very clear requirements
  - Be dependable and perform
  - Reuse when possible

Specification

- Also known as “requirements engineering”
- The process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.
- Sometimes preceded with a feasibility or marketing study
- Output is a requirements document that specifies a system satisfying stakeholder requirements.
  - Customers need a high-level statement of requirements
  - System developers need a more detailed system specification

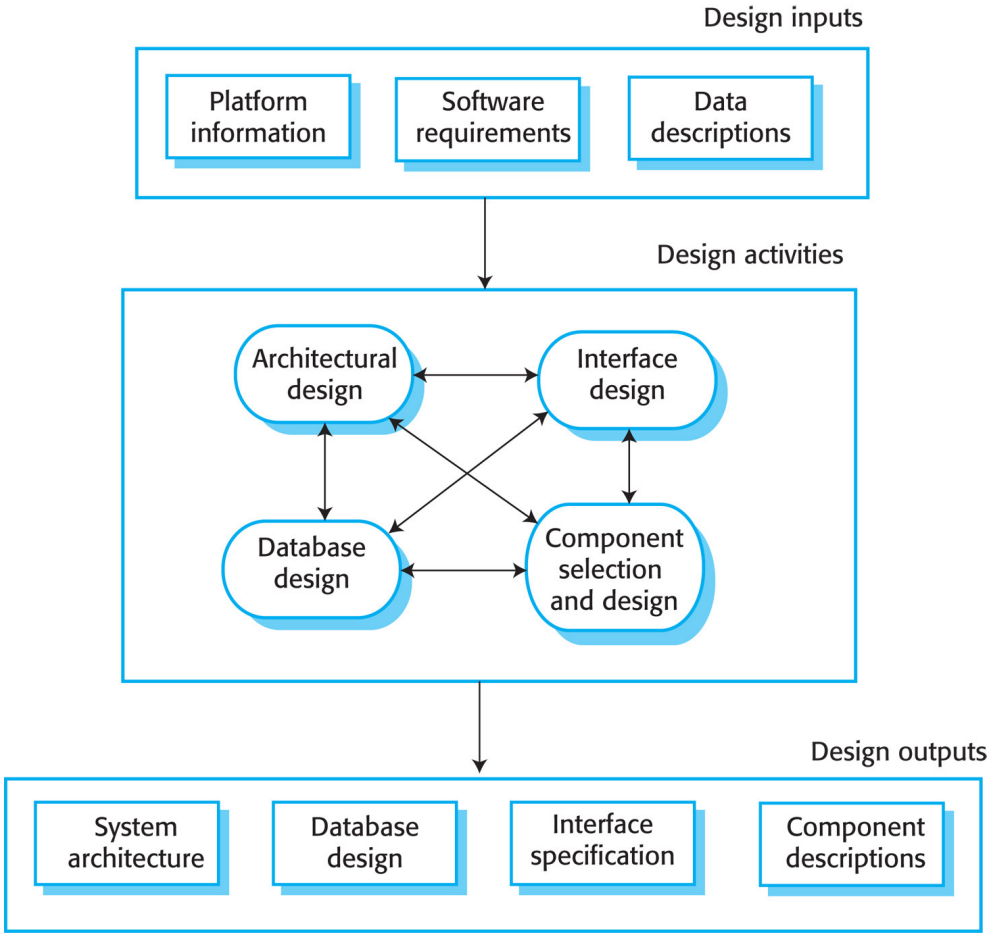
Figure 2.4 The requirements engineering process



Development

- The process of developing an executable system for delivery to the customer.
  - Design – description of the structure of the software to be implemented, the data models, the interfaces and algorithms.
  - Implementation - the actual programming / coding of the system.
- Design generally follows a structured input-> activity-> output model
- The Design outputs are the inputs to Implementation
- Programmers generally do their own “unit test” before software validation phase

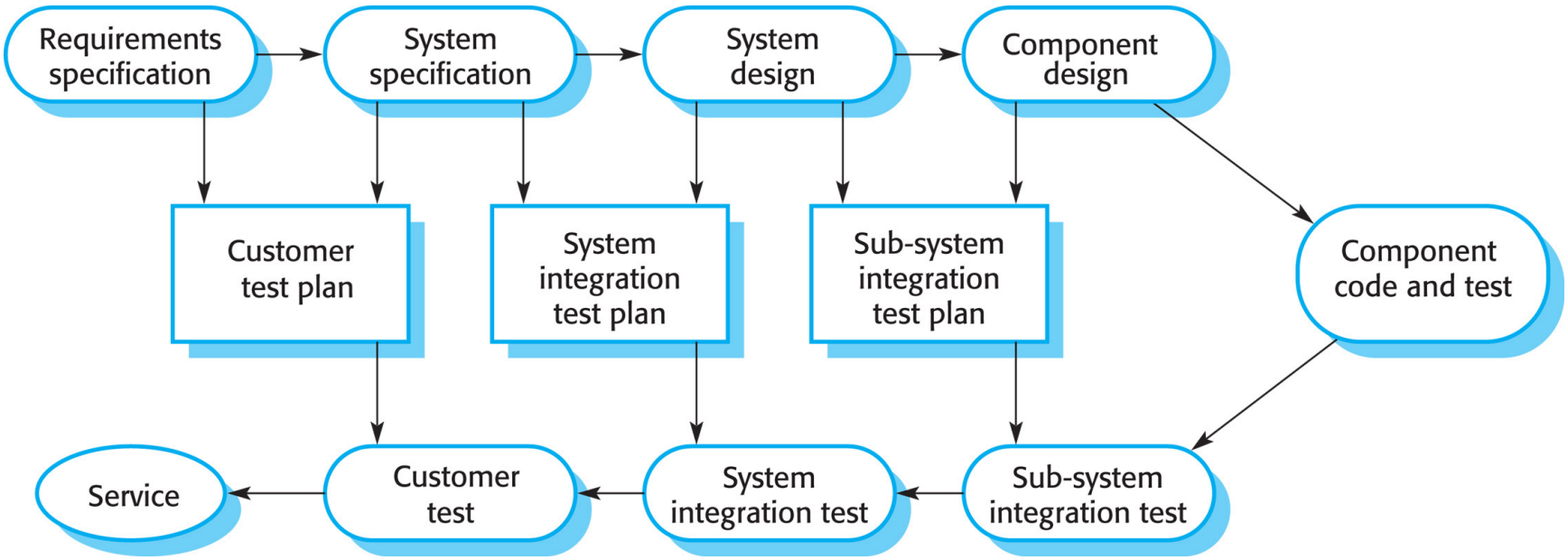
Figure 2.5 A general model of the design process



Validation

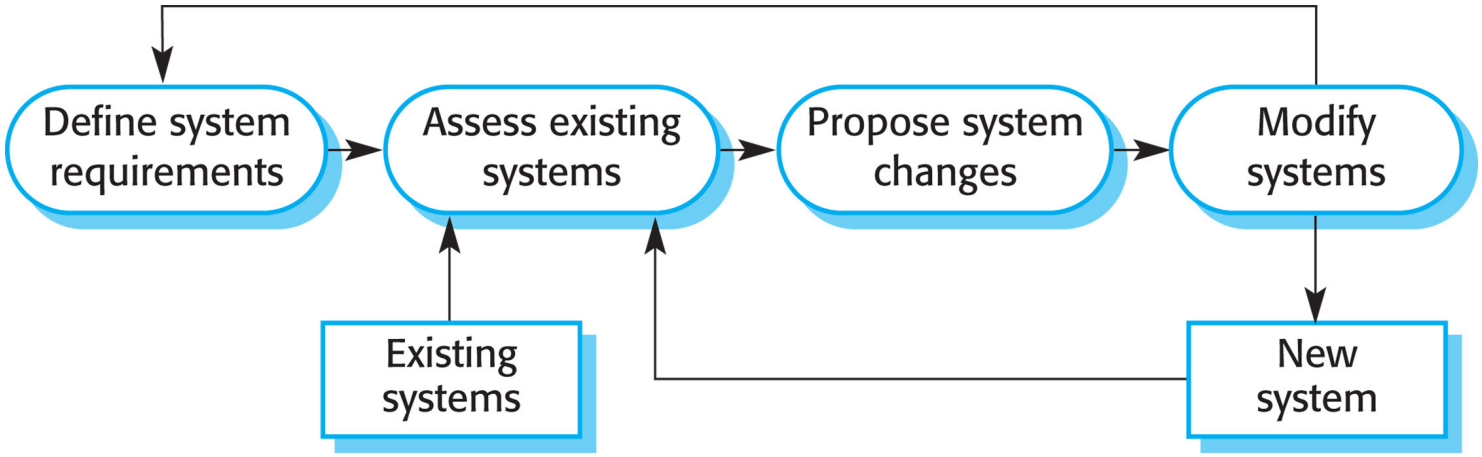
- Also known as “verification and validation (V&V)”
- Assures that a system both conforms to its specification and meets the expectations of the customer (dependable, usable, performs, etc)
- Testing occurs in 3 phases
  - Component – independent testing of each component separately. Test automation tools commonly used.
  - System – integration and testing of the system as a whole. Show that system meets specifications
  - Customer – ensures key requirements met, including performance and usability.
- “Beta testing” is a form of customer testing when a first “release candidate” is tested in the field by select customers

Figure 2.7 Testing phases in a plan-driven software process



Evolution

- Traditionally, there has been a split between people and processes for developing software and maintaining software
- Due to the evolution of both processes (“agile”) and technology (“cloud”), this is changing to more of a constant evolutionary process.



Software Processes

- For each phase of the lifecycle, there are processes which:
  - Have defined outcomes
  - Have defined roles and responsibilities for the team
  - Have pre- and post-conditions
- Each process needs to ensure that the 4 essential attributes of good software are maintained:
  - Acceptability – acceptable to the target user, usable
  - Dependability/Security – should not cause physical or economic damage in the event of a failure
  - Efficiency – don't make wasteful use of system resources
  - Maintainability – can evolve and be understood by others

Figure 1.2 Essential attributes of good software

Product characteristic	Description
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

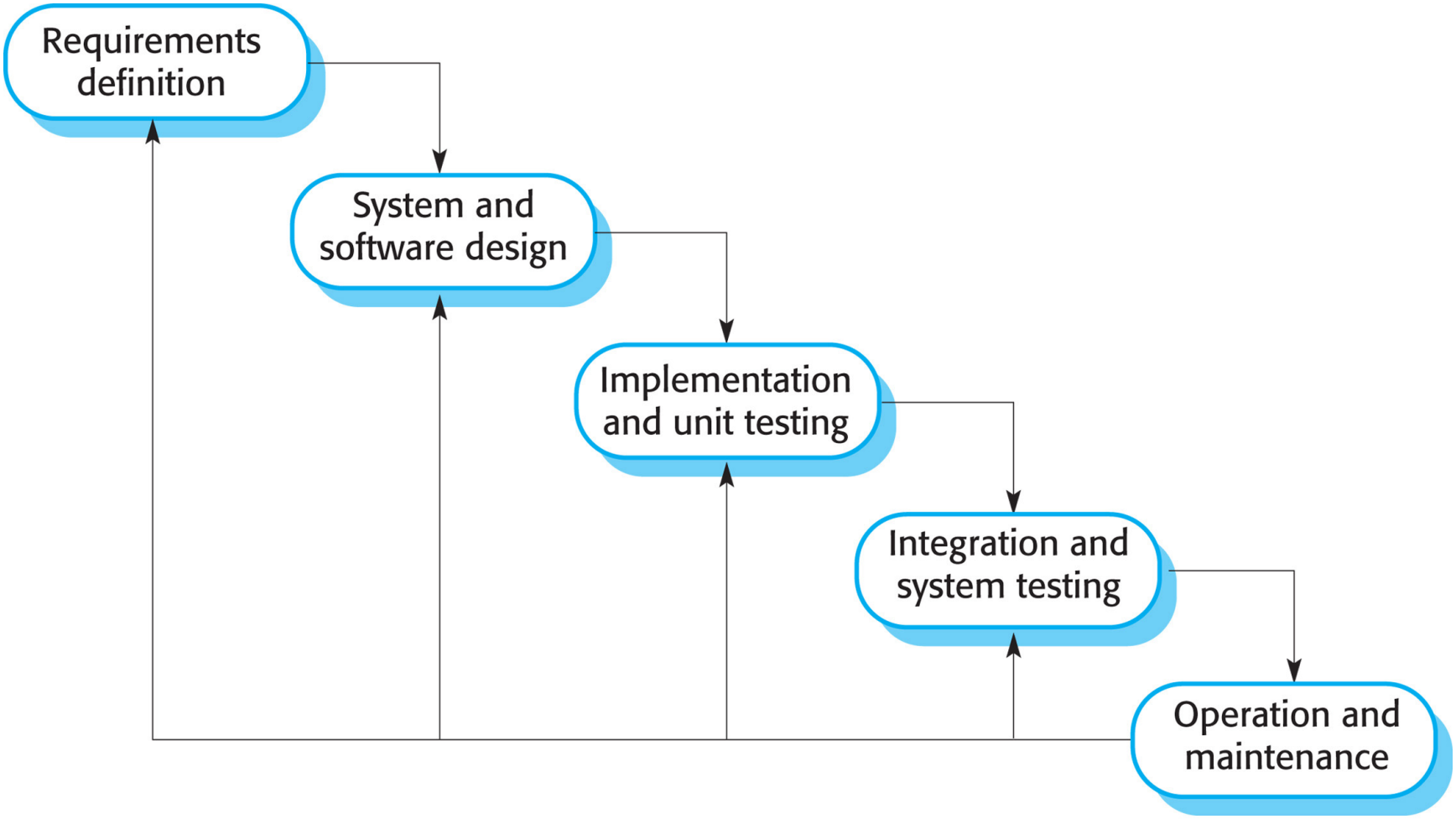
Software Process Models

- A Software Process Model is a simplified representation of a software process.
- Each process model represents a process from a particular perspective
- There are 2 major types of general process models:
  - Waterfall model – Treats the 4 major process activities as separate, serial phases, each with well defined sub-processes
  - Incremental development model – various ways to interleave the 4 major process activities in order to create a more efficient overall process

The waterfall model

- The first published model for software development
- Represents software development as a cascading set of 5 stages:
  - Requirements definition – system's services, constraints and goals are established by consultation with system users.
  - System and software design – hardware/software requirements, system architecture, components and relationships
  - Implementation and unit testing – instantiate the design in code units. Test each unit to ensure it meets specification
  - Integration and system testing – Integration of program units into a system and tested vs. requirements
  - Operation and maintenance – on-going support and enhancements
- One phase does not start until exit criteria of previous phase are met

Figure 2.1 The waterfall model

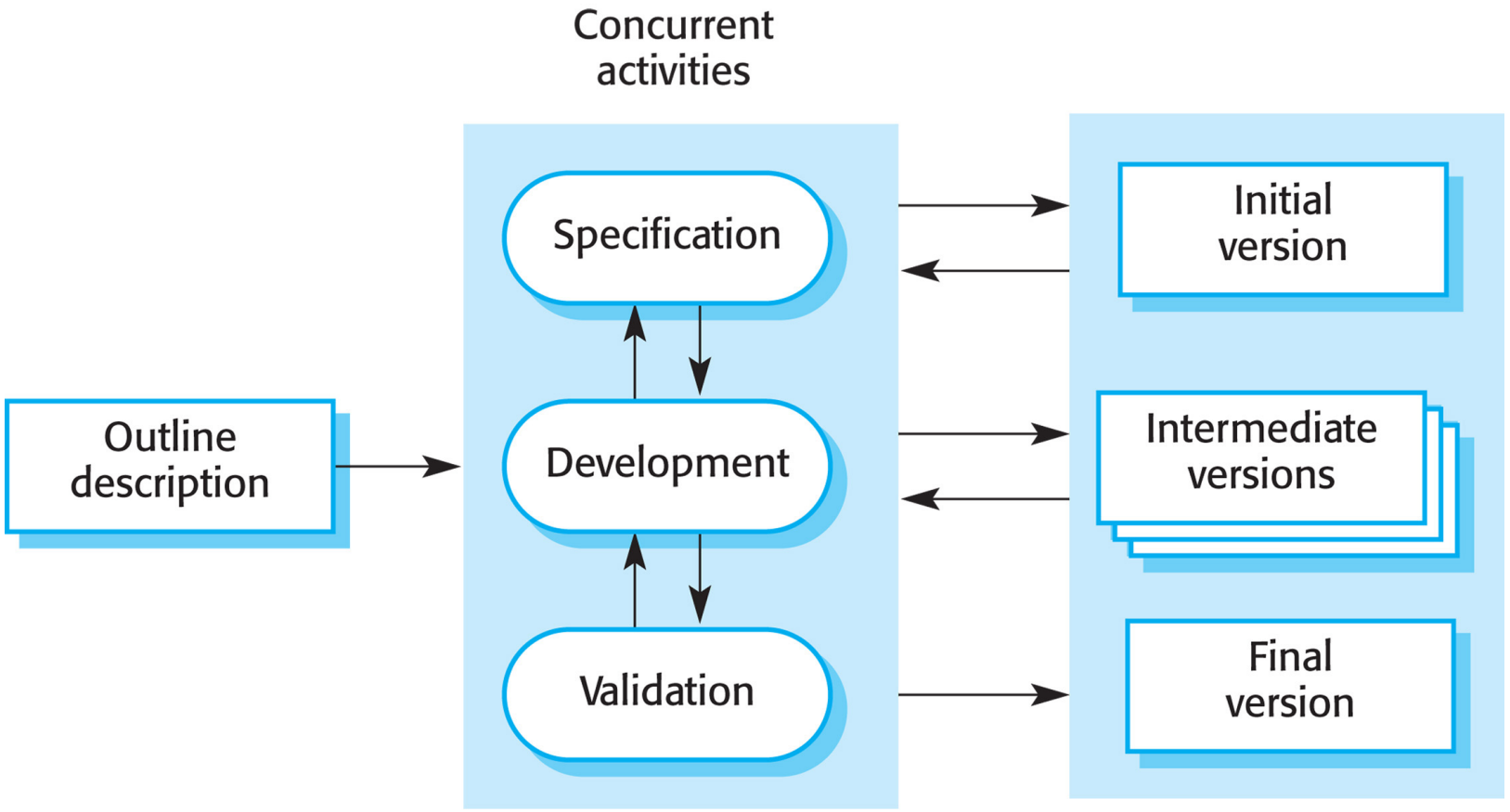


Incremental development model

- Based on the idea of developing an initial implementation, getting feedback, and evolving software through iteration.
- Specification, development and validation activities are interleaved
- Best when requirements are likely to change and the team is able to interact easily across all phases
- "Agile Development" is the preeminent example of an incremental development approach

Advantages	Disadvantages
Cost of implementing requirement changes is lowered	System structure tends to degrade as new increments are added.
Easier to get feedback on running code	The process is not visible
Early delivery of working code is possible	Very difficult for large complex systems

Figure 2.2 Incremental development



CASE STUDY 1 - Mentcare: A patient information system for mental health care

- Throughout the text book there will be a number of case studies used as examples.
- The first one is “Mentcare,” a patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

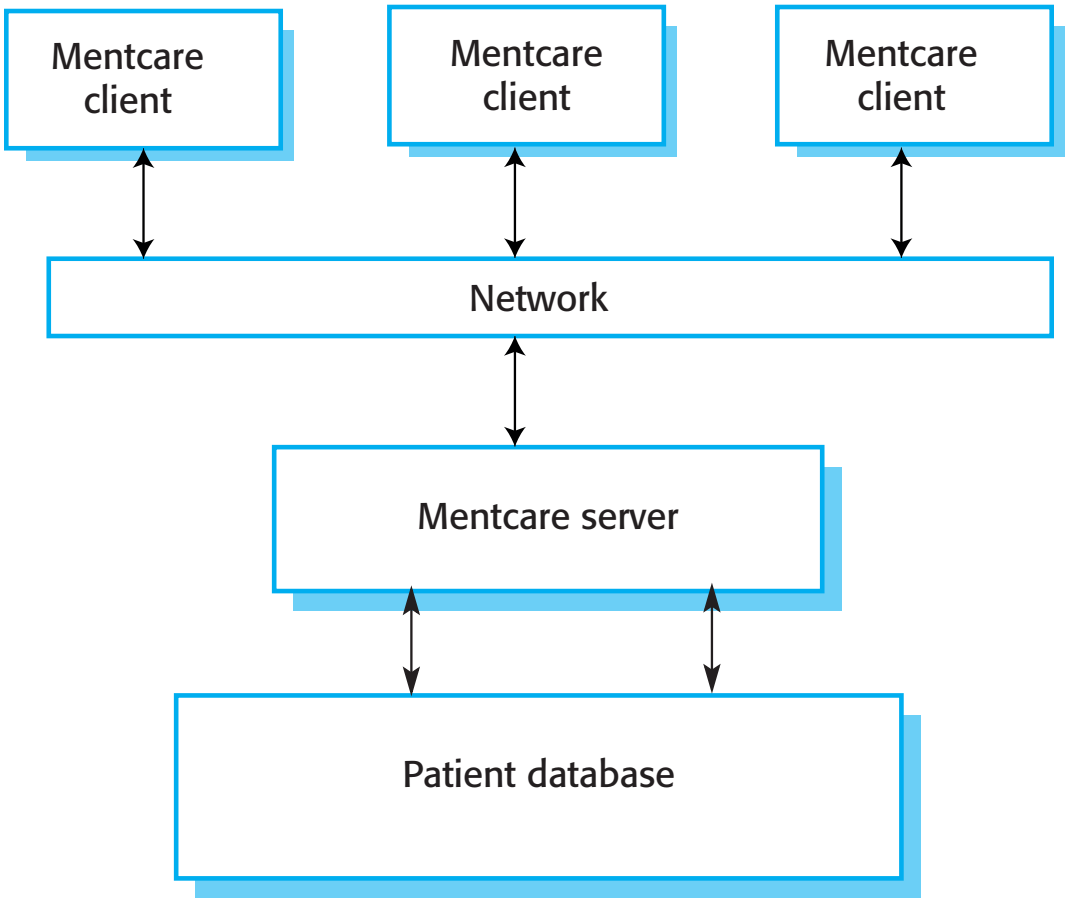
Mentcare

- Mentcare is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

Mentcare goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

The organization of the Mentcare system



Key features of the Mentcare system

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

Mentcare system concerns

- Privacy
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- Safety
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

REQUIREMENTS
TOPICS COVERED

- Requirements Engineering Overview
- Functional and non-Functional Requirements
- Requirements engineering processes
  - Requirements elicitation
  - Requirements specification
  - Requirements validation
- Requirements change

Requirements Engineering Overview

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

What is a requirement?

- 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 inevitable as requirements may serve a 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 abstraction (Davis)

"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system."

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

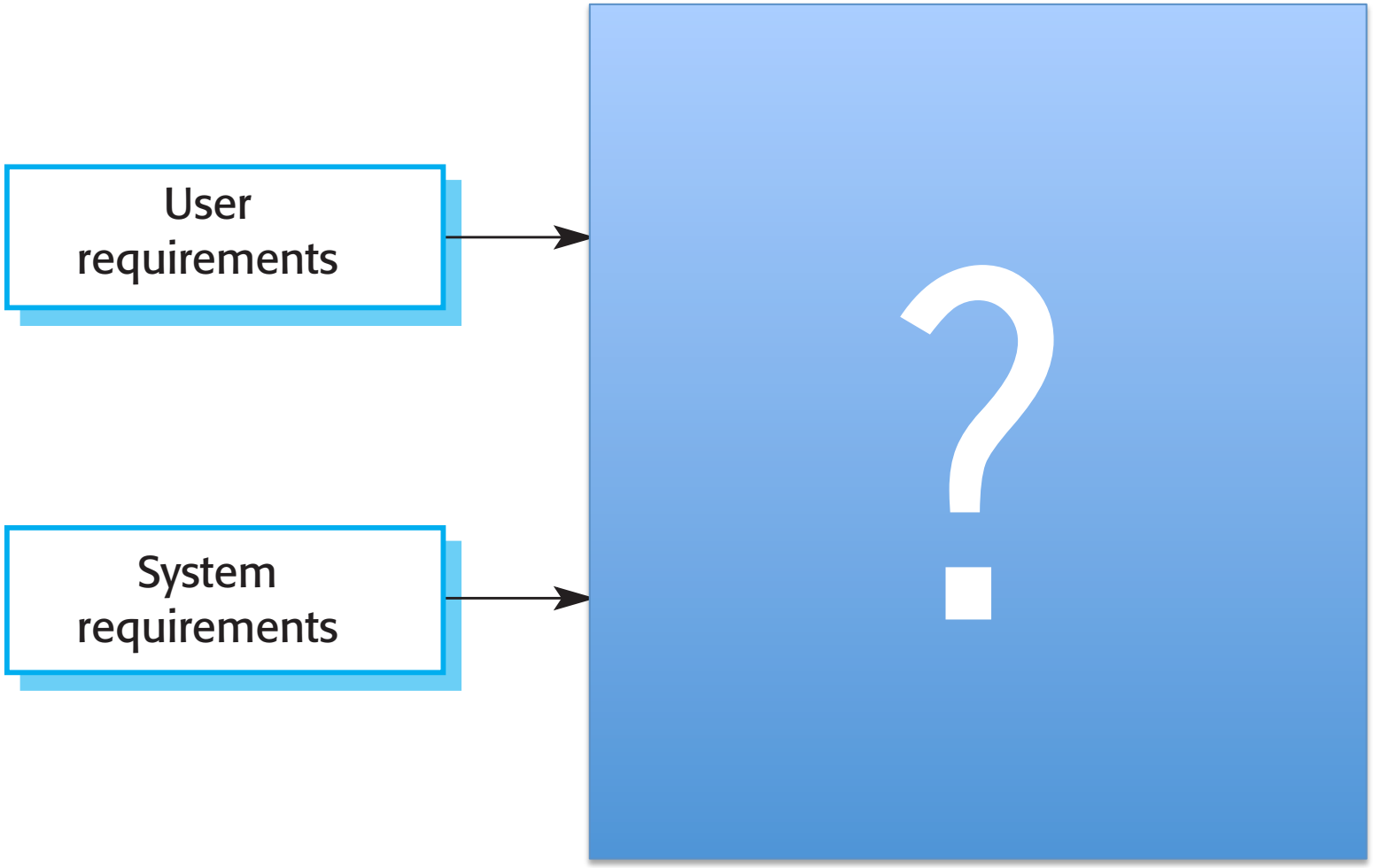
User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Readers of different types of requirements specification



System stakeholders

- *Any person or organization who is affected by the system in some way and so who has a legitimate interest*
- Stakeholder types
  - End users
  - System managers
  - System owners
  - External stakeholders

Stakeholders in the Mentcare system

- Patients whose information is recorded in the system.
- Doctors who are responsible for assessing and treating patients.
- Nurses who coordinate the consultations with doctors and administer some treatments.
- Medical receptionists who manage patients' appointments.
- IT staff who are responsible for installing and maintaining the system.

Stakeholders in the Mentcare system

- A medical ethics manager who must ensure that the system meets current ethical guidelines for patient care.
- Health care managers who obtain management information from the system.
- Medical records staff who are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

Agile methods and requirements

- Many agile methods argue that producing detailed system requirements is a waste of time as requirements change so quickly.
- The requirements document is therefore always out of date.
- Agile methods usually use incremental requirements engineering and may express requirements as ‘user stories’ (discussed in Chapter 3).
- This is practical for business systems but problematic for systems that require pre-delivery analysis (e.g. critical systems) or systems developed by several teams.

- Functional and non-functional requirements

Functional and non-functional requirements

- 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
  - Not directly concerned with the specific services delivered by the system to its users
  - 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.
  - Examples include Reliability, Response Time and Memory Use

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Mentcare system: functional requirements

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Requirements imprecision

- Problems arise when functional requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘search’ in requirement 1
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

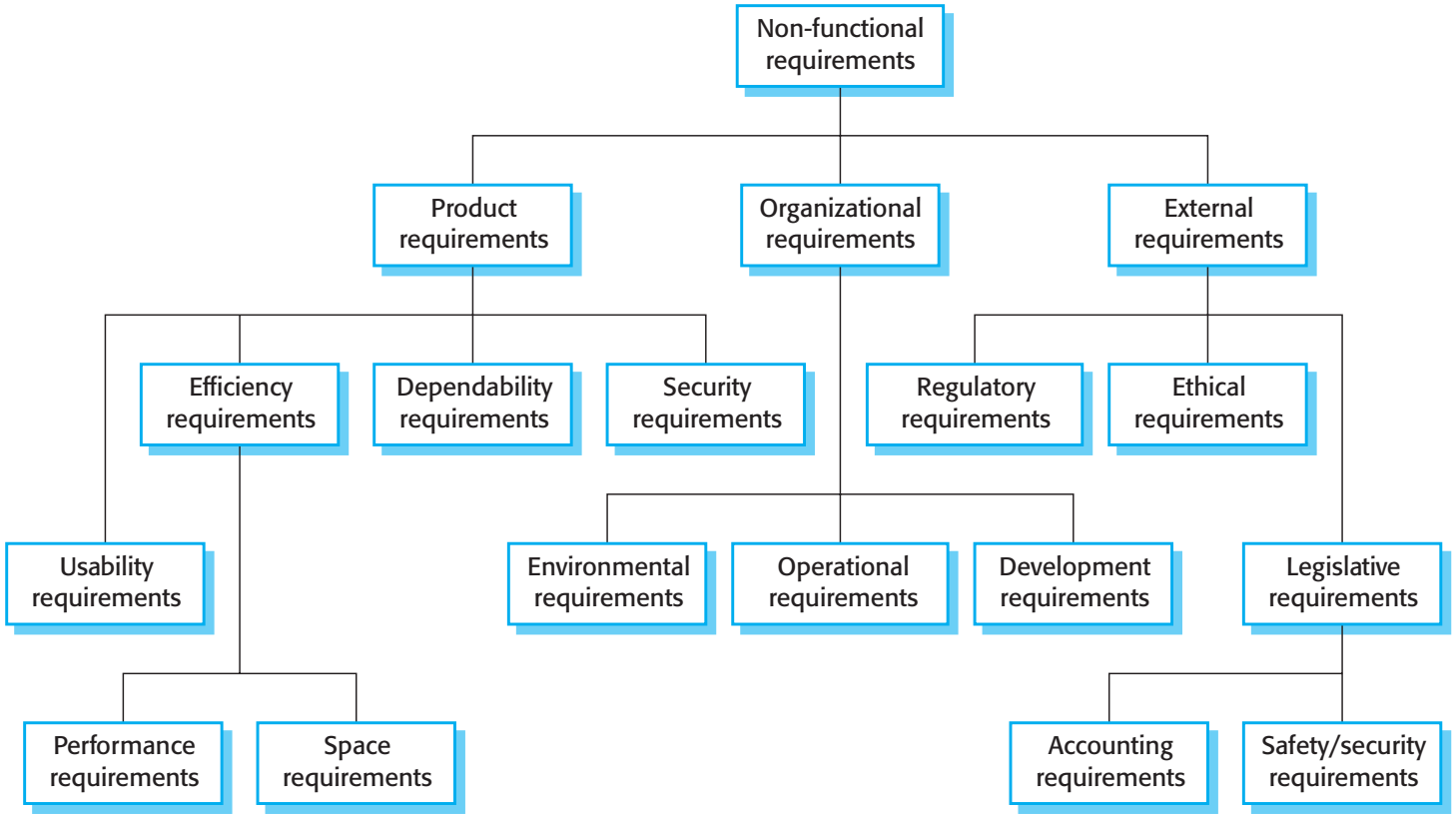
Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
- Complete
  - They should include descriptions of all facilities required.
- Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of nonfunctional requirement



Non-functional requirements implementation

- 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

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organizational 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.

Examples of nonfunctional requirements in the Mentcare system

Product requirement
The Mentcare 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 Mentcare 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

- 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

- 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 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

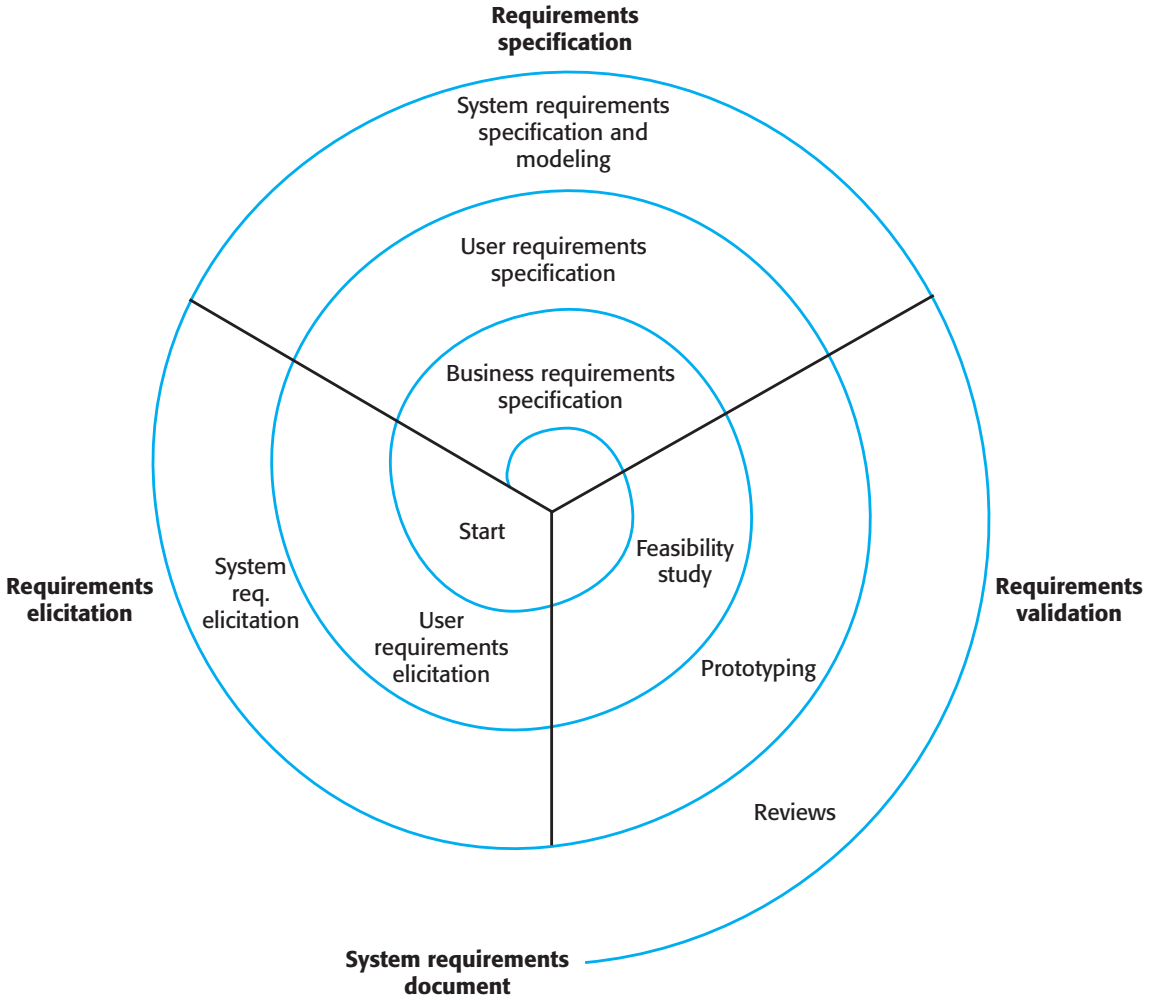
Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart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

- Requirements engineering processes

Requirements engineering processes

- The processes used for Requirements Engineering (RE) vary widely depending on the application domain, the people involved and the organization developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

A spiral view of the requirements engineering process



Requirements elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

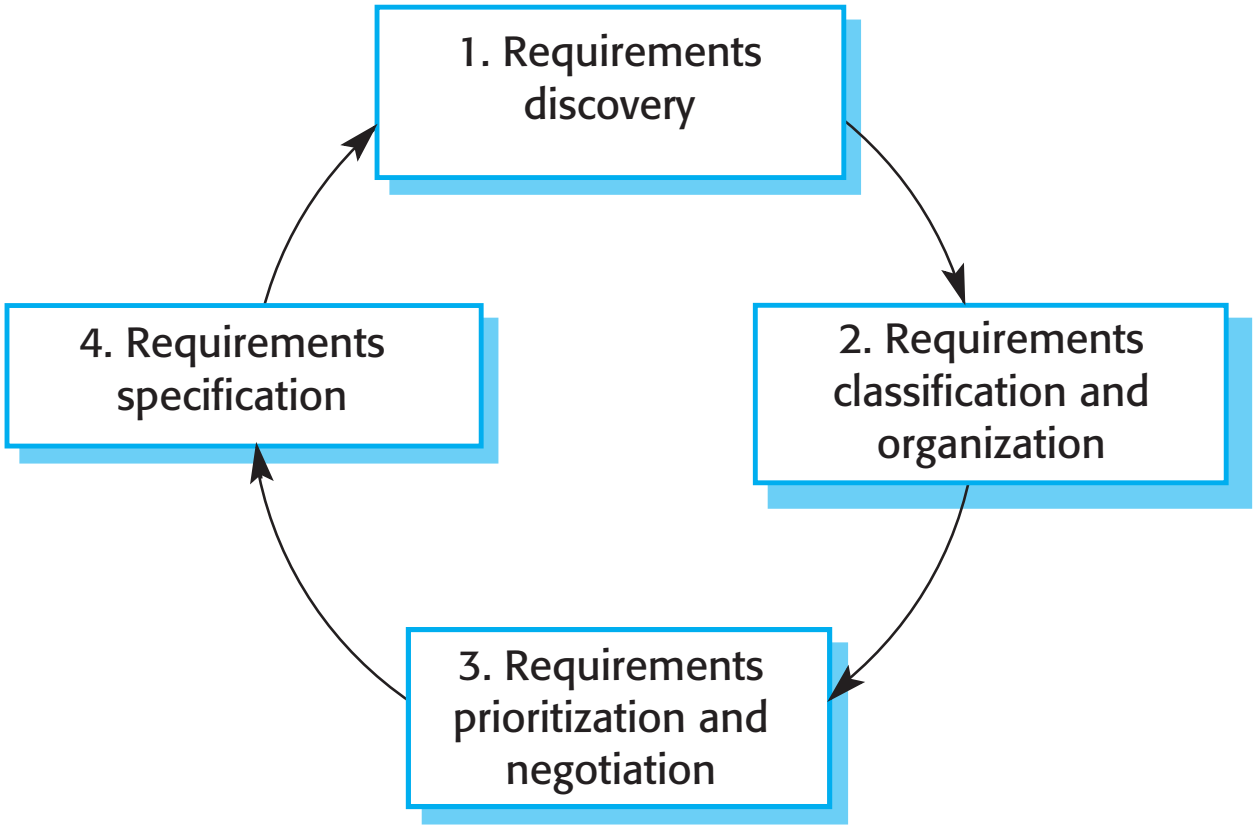
Requirements elicitation

- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- Stages include:
  - Requirements discovery,
  - Requirements classification and organization,
  - Requirements prioritization and negotiation,
  - Requirements specification.

Problems of requirements elicitation

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

The requirements elicitation and analysis process



Process activities

1. Requirements discovery
   - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
2. Requirements classification and organization
   - Groups related requirements and organises them into coherent clusters.
3. Prioritization and negotiation
   - Prioritising requirements and resolving requirements conflicts.
4. Requirements specification
   - Requirements are documented and input into the next round of the spiral.

Requirements discovery

- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Interaction is with system stakeholders from managers to external regulators.
- Systems normally have a range of stakeholders.

Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
  - Closed interviews based on pre-determined list of questions
  - Open interviews where various issues are explored with stakeholders.
- Effective interviewing
  - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
  - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviewers need to be open-minded without pre-conceived ideas of what the system should do
- You need to prompt the use to talk about the system by suggesting requirements rather than simply asking them what they want.

Ethnography

- A social scientist spends a considerable time observing and analysing how people actually work.
- People do not have to explain or articulate their work.
- Social and organizational factors of importance may be observed.
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

Scope of ethnography

- Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.
- Requirements that are derived from cooperation and awareness of other people's activities.
  - Awareness of what other people are doing leads to changes in the ways in which we do things.
- Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Stories and scenarios

- Scenarios and user stories are real-life examples of how a system can be used.
- Stories and scenarios are a description of how a system may be used for a particular task.
- Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.
- See example in book (iLearn) for more detail

Requirements specification

Requirements specification

- The process of writing down the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
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. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

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 may be a domain requirement.
  - This may be the consequence of a regulatory requirement.

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

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.
- Include an explanation (rationale) of why a requirement is necessary.

Problems with natural language

- Lack of clarity
  - Precision is difficult without making the document difficult to read.
- Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
- Requirements amalgamation
  - Several different requirements may be expressed together.

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

A structured specification of a requirement for an insulin pump

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—the dose in insulin to be delivered.

Destination Main control loop.

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

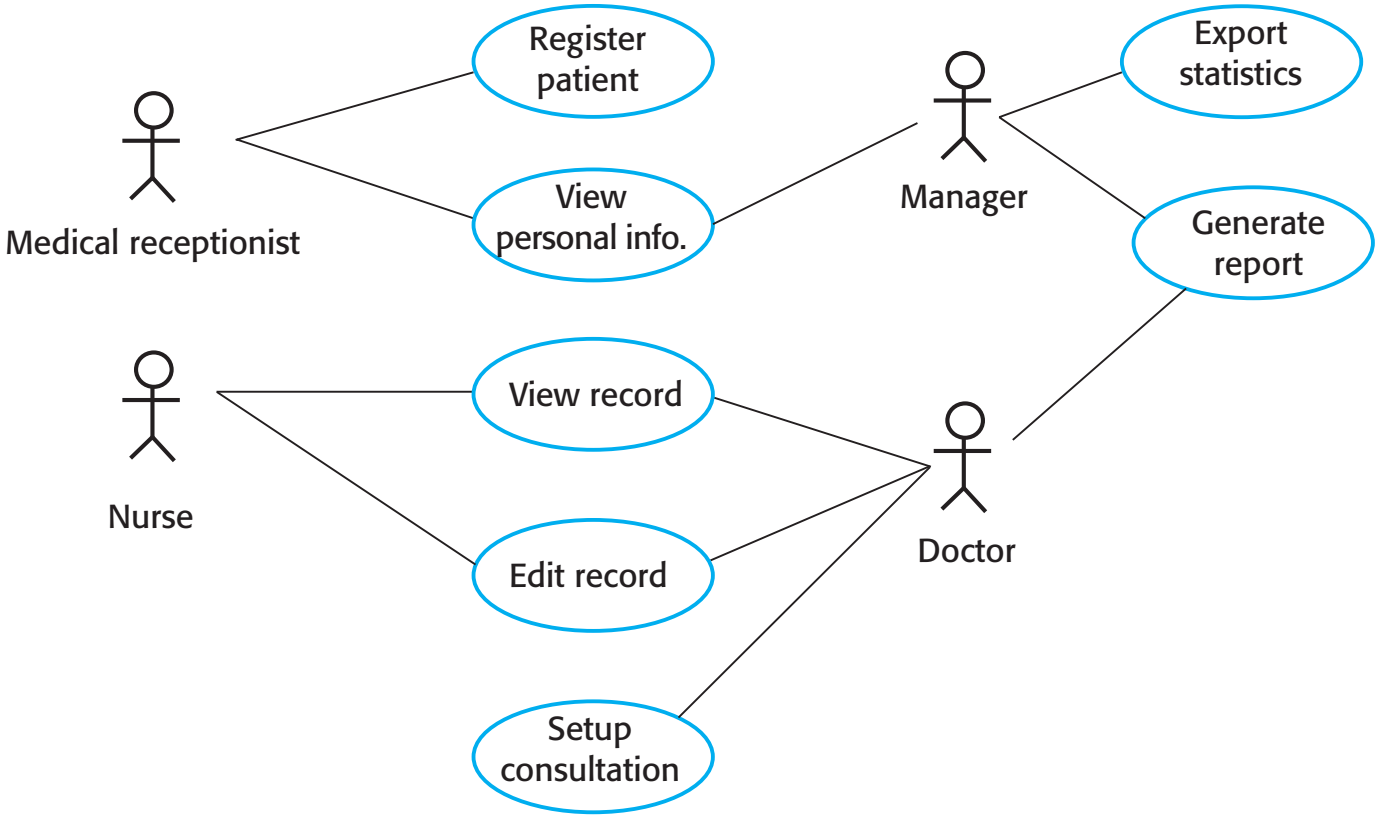
Tabular specification of computation for an insulin pump

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

Use cases

- Use-cases are scenarios of how users interact with the system
- Use cases 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.
- High-level graphical model supplemented by more detailed tabular description (see Chapter 5).
- We will discuss more when we get to Unified Modeling Language (UML) sequence diagrams, which may be used to add detail to use-cases by showing the sequence of event processing in the system.

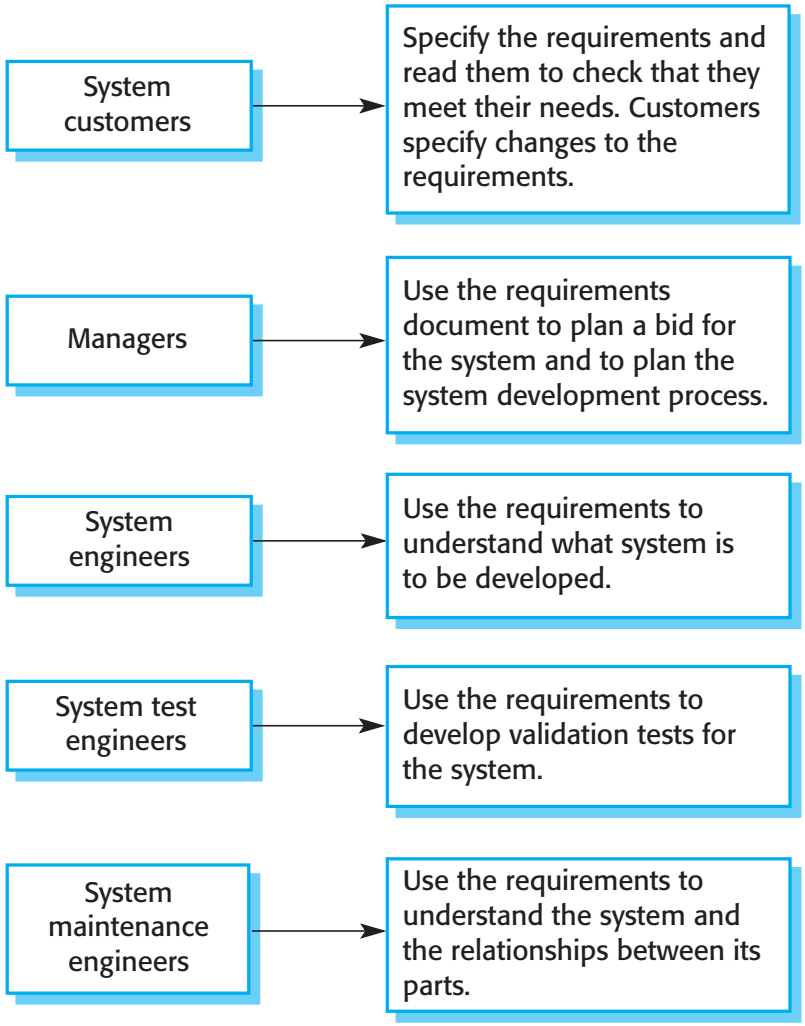
Use cases for the Mentcare system



The software requirements document

- The software 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 set of WHAT the system should do rather than HOW it should do it.

Users of a requirements document



Requirements document variability

- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Requirements validation

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 checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2.
- 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.

Review checks

- Verifiability
  - Is the requirement realistically testable?
- Comprehensibility
  - Is the requirement properly understood?
- Traceability
  - Is the origin of the requirement clearly stated?
- Adaptability
  - Can the requirement be changed without a large impact on other requirements?

Requirements change

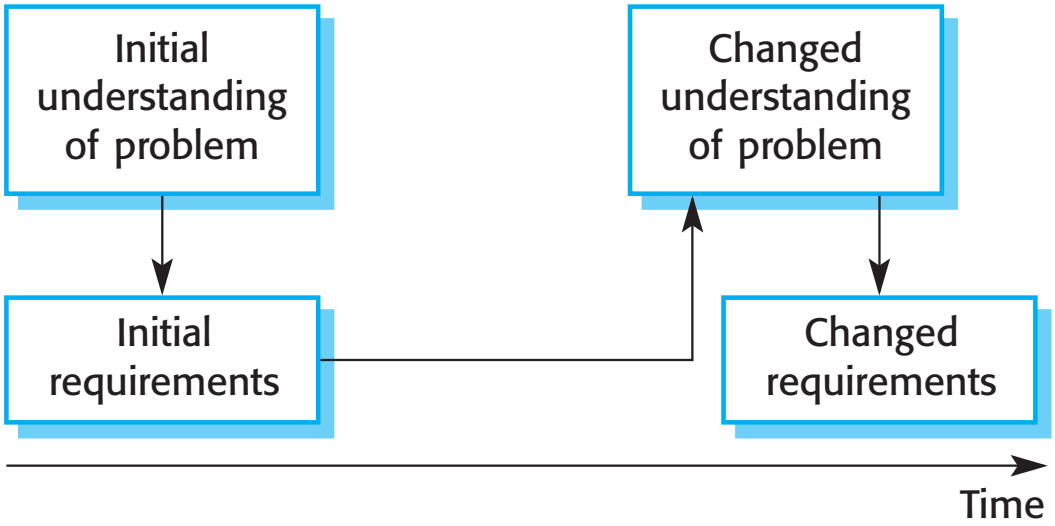
Changing requirements

- The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
  - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

Requirements evolution



Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

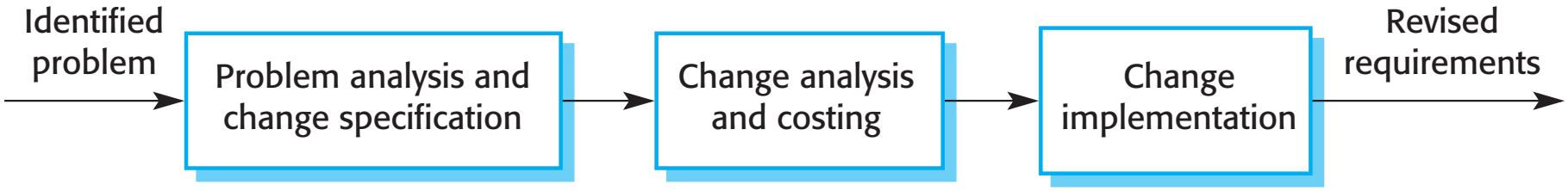
Requirements management planning

- Establishes the level of requirements management detail that is required.
- Requirements management decisions:
  - *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
  - *A change management process* This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
  - *Traceability policies* These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
  - *Tool support* Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements change management

- Deciding if a requirements change should be accepted
  - *Problem analysis and change specification*
    - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
  - *Change analysis and costing*
    - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
  - Change implementation
    - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

Requirements change management



Key points

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Key points

- The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.
- Requirements elicitation is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.
- You can use a range of techniques for requirements elicitation including interviews and ethnography. User stories and scenarios may be used to facilitate discussions.

Key points

- Requirements specification is the process of formally documenting the user and system requirements and creating a software requirements document.
- The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.

Key points

- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.
- Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.

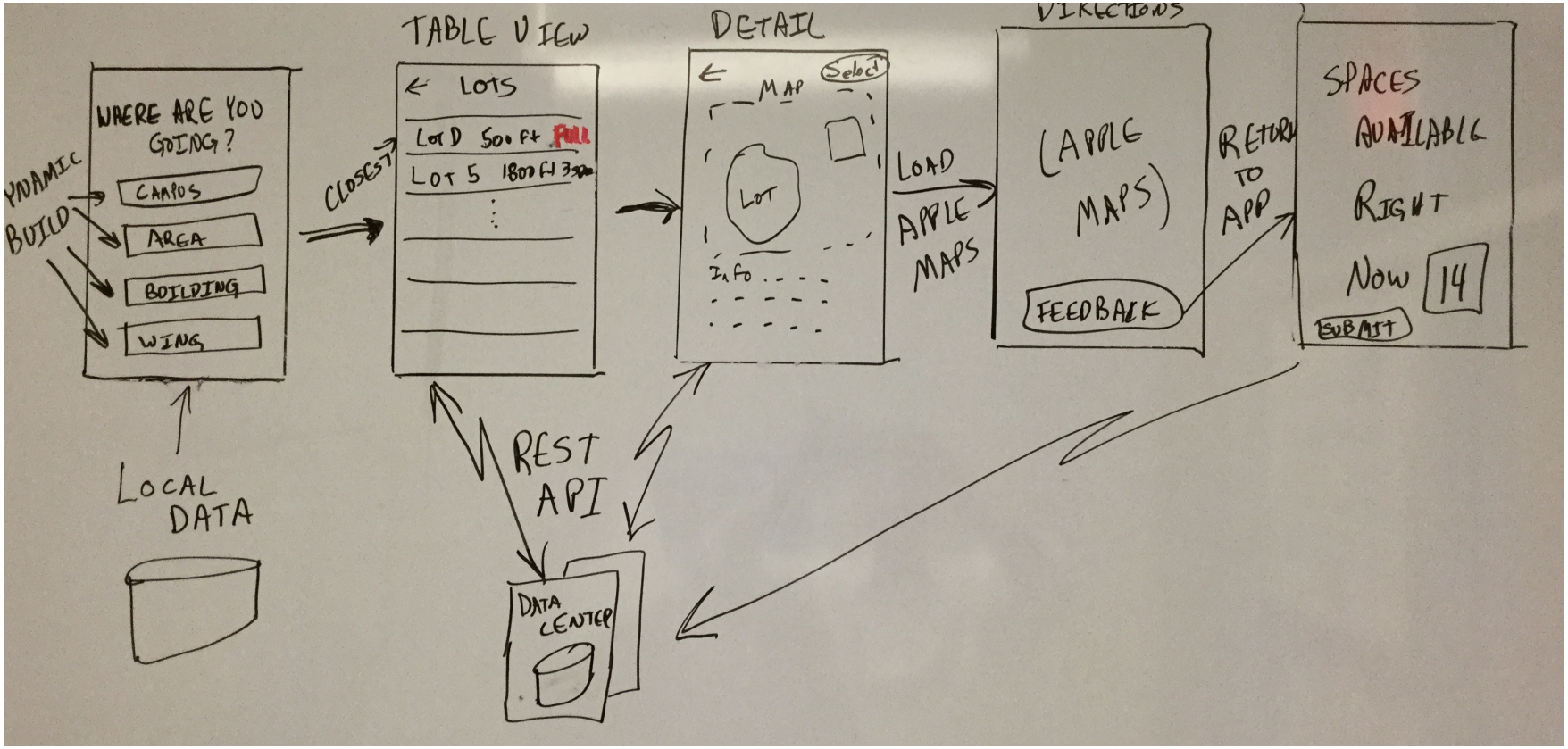
Requirements EXERCISE

- I represent the client who needs an app created.
- The app will be used by Duke students, faculty and staff to find a parking spot.
- I have prepared a few slides to explain what I want, but like most clients the requirements can be incomplete and potentially conflicting.
- As the development team, you all must bring clarity and completeness to my requirements by **asking questions** and **taking careful notes.**
- Your notes will form the basis of your homework assignment.

"Duke Parking Finder"

- Problem: Too difficult to find the parking lots, let alone a parking space, on campus.
- Solution: An app that has all the latest information on where the parking lots are and, if applicable, where there is available parking
- Approach:
  - Create a database of all parking lots on campus
    - GPS location
    - Street address
    - Number of spots
    - Types of parking (which permits, # of visitor spots, etc)
    - Available spots (if applicable)
  - Query users for where their class/meeting is and list closest parking lots.
    - Allow drill down into each lot
    - Once user selects a lot, provide Apple Map guidance
  - Allow user feedback where real-time data can be fed back into system
- Other – need to use the Duke app store, follow Duke security model. Must have some function even if not connected. Needs to be intuitive so no training/user guide required. Needs to use the Duke private cloud infrastructure.

High Level Architecture for Duke Parking Finder



About Week 3 – February 3rd

- Homework Assignments – 50 point max grade / 43 point base grade.
  - Create a Requirements Document for Parking App – 40 /34 points
    - Submit on Sakai a .pdf document entitled "Duke Parking App Requirements Document".
    - The document should contained itemized / categorized requirements for the Parking app.
    - Include User Requirements (Functional and Non-Functional) as well as System Requirements.
    - I put a simple example .doc on Sakai but feel free to expand on this / create your own format (more sections, use structure, tabular, use cases, etc)
    - Homework will be graded on organization, completeness, clarity and granularity.
    - This is an INDIVIDUAL assignment – do you own work.
  - Submit your team members and idea(s) for project – 10 / 9 points
    - Inline in Sakai put your team members, team name and project choice / ideas
    - If you don't have teammates in mind and want to request a team, use Piazza to post what your skills and interest are for a project. OR
    - Just tell me that you want to be put on a team and I will place you.
    - Think of an idea for your semester project or prioritize from the list given and include
- Reading Assignment
  - Software Engineering, 10th Edition, Ian Sommerville
    - Chapter 3 – "Agile software development"