Software Requirements

Minsoo Ryu

Hanyang University

Topics covered

- What are requirements?
- Classifications of Requirements
  - Functional and Non-Functional Requirements
  - User and System Requirements
- Requirements Specification Languages
- Guidelines for Requirements Specification

What Are Requirements?

- The requirements are the descriptions of the services provided by the system and its operational constraints

- Requirements engineering is the process of finding out, analyzing, documenting, and checking these services and constraints

Topics covered

- What is Requirements Engineering?
- Classifications of Requirements
  - Functional and Non-Functional Requirements
  - User and System Requirements
- Requirements Specification Languages
- Guidelines for Requirements Specification

Functional Requirements

- The functional requirements describe what the system should do
  - 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 but functional system requirements should describe the system services in detail.

Non-functional Requirements

- Non-functional requirements are requirements that are not directly concerned with the specific functions delivered by the system
  - They may relate to emergent system properties such as reliability, response time, and store occupancy
  - Alternatively they may define constraints on the system such as the capabilities of I/O devices and the data representations used in system interfaces (performance and compatibility)

- Non-functional requirements may be more critical than functional requirements
  - If these are not met, the system is useless

Functional and Non-functional Requirements

- In reality, the distinction between different types of requirements is not as clear-cut as the simple definitions suggest
  - A user requirement concerned with security may appear to be a non-functional requirement
  - However, when developed in more detail, this requirement may generate other requirements that are clearly functional such as the need to include user authentication facilities in the system

A Problem with Non-functional Requirements

- A common problem with non-functional requirements is that they can be difficult to verify
  - Users or customers often state these requirements as general goals such as ease of use, the ability of the system to recover from failure or rapid user response

- Whenever possible, you should write non-functional requirements quantitatively so that they can be objectively tested

User and System Requirements

- By 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 predefined
  - 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

User Requirements

- User requirements should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge
  - They should only specify the external behaviour of the system and should avoid , as far as possible, system design characteristics
  - You should not use software jargon, structured notations or formal notations, or describe the requirement by describing the system implementation
  - User requirements are defined using natural language, tables and diagrams as these can be understood by all users

System Requirements

- System requirements are expanded versions of the user requirements
  - They may be used as part of the contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system
  - Ideally, the system requirements should simply describe the external behaviour of the system and its operational constraints
  - They should not be concerned with how the system should be designed or implemented

- However, at the level of detail required to completely specify a complex software system, it is impossible, in practice, to exclude all design information

Topics covered

- What is Requirements Engineering?
- Classifications of Requirements
  - Functional and Non-Functional Requirements
  - User and System Requirements
- Requirements Specification Languages
- Guidelines for Requirements Specification

Requirements Specification Languages

- Natural language-based specification
- Form-based specification
- Tabular specification
- Graphical specification
- Formal specification

Natural Language-based Specification

- Natural language is often used to write system requirements specification
  - However, because system requirements are more detailed than user requirements, natural language specifications can be confusing and hard to understand

- We can write system requirements in more specialized notations
  - Structured natural language
  - Design description languages
  - Graphical notation
  - Mathematical specifications

Form-based Specification

- When a standard form is used for requirements specification, the following information should be included
  - 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.

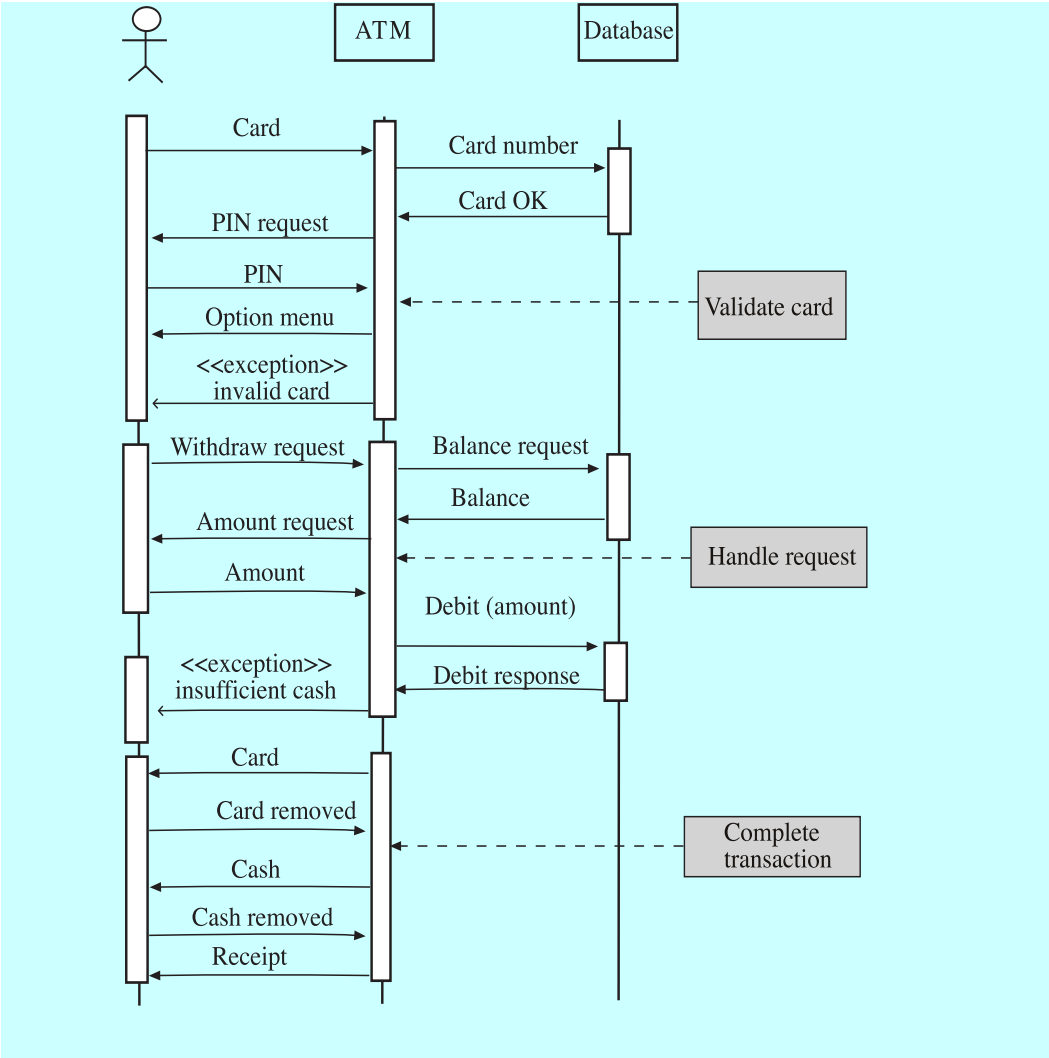
Form-based Specification (Insulin Pump)

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 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

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

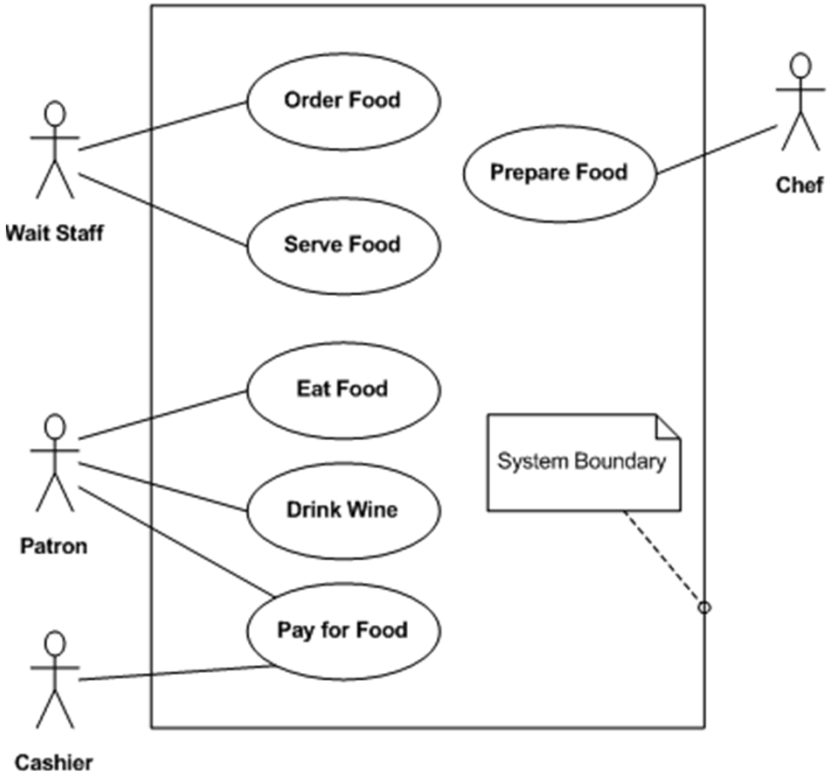
Graphical Specification (ATM withdrawal)



Formal Specification Languages: UML

- **UML (Unified Modeling Languages)**
  - **Visual and object-oriented**
  - **Diagrams + natural languages**

Use Case Diagram



Formal Specification Languages: VDM

- **The Vienna Development Method (VDM) is one of the longest-established Formal Methods for the development of computer-based systems**
  - **Originating in work done at IBM's Vienna Laboratory in the 1970s, it has grown to include a group of techniques and tools based on a formal specification language - the VDM Specification Language (VDM-SL)**

- **Features like programming languages**
  - **Various types: numeric, character, token and quote types**
    - Type constructors are used to build more structured data types
  - **Collections and operators: sets, mappings, and sequences**

Formal Specification Languages: VDM

- Example 1:
  - SQRT(x:nat)r:real
  - post r*r = n and r>=0
- Example 2:
  - SQRTP(x:real)r:real
  - pre x >=0
  - post r*r = n and r>=0

Topics covered

- What is Requirements Engineering?
- Classifications of Requirements
  - Functional and Non-Functional Requirements
  - User and System Requirements
- Requirements Specification Languages
- Guidelines for Requirements Specification

Problems with Natural Language

- **Lack of clarity**
  - It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read
- **Requirements confusion**
  - Functional requirements, non-functional requirements, system goals, and design information may not be clearly distinguished
- **Requirements amalgamation**
  - Several different requirements may be expressed together

Unambiguity

- **Imprecision in the requirements specification is the cause of many software engineering problems**
  - **Ambiguous requirements may be interpreted in different ways by developers and users**

- **Consider the term ‘appropriate viewers’**
  - **Customer intention - special purpose viewer for each different document type**
  - **Developer interpretation - Provide a text viewer that shows the contents of the document**
    - The requirement is worded ambiguously
    - A developer might claim that the requirement had been met

Completeness and Consistency

- **In principle, requirements should be both complete and consistent**

- **Complete**
  - **They should include descriptions of all facilities required.**
  - **MECE (mutually exclusive, collectively exhaustive)**
    - No overlaps, no gaps

- **Consistent**
  - **There should be no conflicts or contradictions in the descriptions of the system facilities**

Testability

- Most requirements should be testable
  - If this is not the case, another verification method should be used instead (e.g. analysis, inspection or review of design)
  - Testable requirements are an important component of validation
- Certain requirements, by their very structure, are not testable
  - These include requirements that say the system shall never or always exhibit a particular property
  - Proper testing would require an infinite testing cycle
- Un-testable non-functional requirements may still be kept as a documentation of customer intent; however they are usually traced to process requirements that are determined to be a practical way of meeting them
  - For example, a non-functional requirement to be free from backdoors may be satisfied by replacing it with a process requirement to use pair programming

Requirements Engineering

Minsoo Ryu

Hanyang University

Topics covered

- Requirements Engineering
- Requirements Elicitation
- Requirements Validation
- Requirements Management

Requirement Engineering

- **The goal of the requirements engineering process is to create and maintain a system requirements document**
- **Three sub-processes**
  - **Elicitation**
    - Discovering requirements (elicitation)
    - Converting the requirements into some standard form (specification)
  - **Validation**
    - Checking that the requirements
  - **Management**
    - Managing requirements changes

Requirements Elicitation

- Requirements elicitation should find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on
  - This may involve a variety of people (stake holders)
  - Stakeholders are the people who will be affected by the system
  - Stakeholders include end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc

Difficulties with Eliciting Stakeholder Requirements

- There are several reasons for the difficulties
  - Stakeholders don't know what they really want
  - Stakeholders express requirements in their own terms and with implicit knowledge of their own work
  - Different stakeholders may have conflicting requirements
  - Political factors may influence the system requirements
  - The economic and business environment in which the analysis takes place is dynamic

Difficulties with Eliciting Stakeholder Requirements

- It is important to determine the following four elements of the project:
  - the purpose, aims and a basic understanding of the problem that requires a computer solution;
  - an understanding of the context and the environment in which the project will take place;
  - an understanding of the impact that the client expects the system to have;
  - an understanding of the motivation and expectations of the client

Requirements Elicitation Techniques

- Interviewing
- Scenarios
- Use-cases
- Ethnography
- Other techniques
  - Questionnaires
  - Focus Groups
  - Apprenticing
  - Brainstorming
  - Document Inspection

Interviewing

- **Formal or informal interviews with stakeholders**
  - **Puts questions to stakeholders**
  - **Requirements are derived from the answers**

- **Two types of interview**
  - **Closed interviews**
    - The stakeholders answer a pre-defined set of questions
  - **Open interviews**
    - There is no pre-defined agenda
    - Explores a range of issues with system stakeholders and hence develops a better understanding of their needs

Interviews in Practice

- Interviews are normally a mix of closed and open interviews
  - Most interviews require some questions to get started and to keep the interview focused on the system to be developed
  - The answers may lead to other issues that are discussed in a less structured way

- Interviews are good for getting an overall understanding of:
  - What stakeholders do
  - How they might interact with the system
  - The difficulties that they face with current systems

Effective Interviewers

- Interviewers should be open-minded, avoid preconceived ideas about the requirements, and willing to listen to stakeholders
  - If the stakeholder comes up with surprising requirements, they are willing to change their mind about the system
- They should prompt the interviewee to start with a question, a requirements proposal, or by suggesting working together on a prototype system
  - Saying to people ‘tell me what you want’ is unlikely to result in useful information
  - Most people find it much easier to talk in a defined context rather than in general terms

Scenarios

- Scenarios are real-life examples of how a system can be used
  - People usually find it easier to relate real-life examples than to abstract descriptions
  - Scenarios can be particularly useful for adding detail to an outline requirements

- A scenario may include;
  - A description of the starting situation
  - A description of the normal flow of events
  - A description of what can go wrong and how that is handled
  - Information about other concurrent activities
  - A description of the state when the scenario finishes

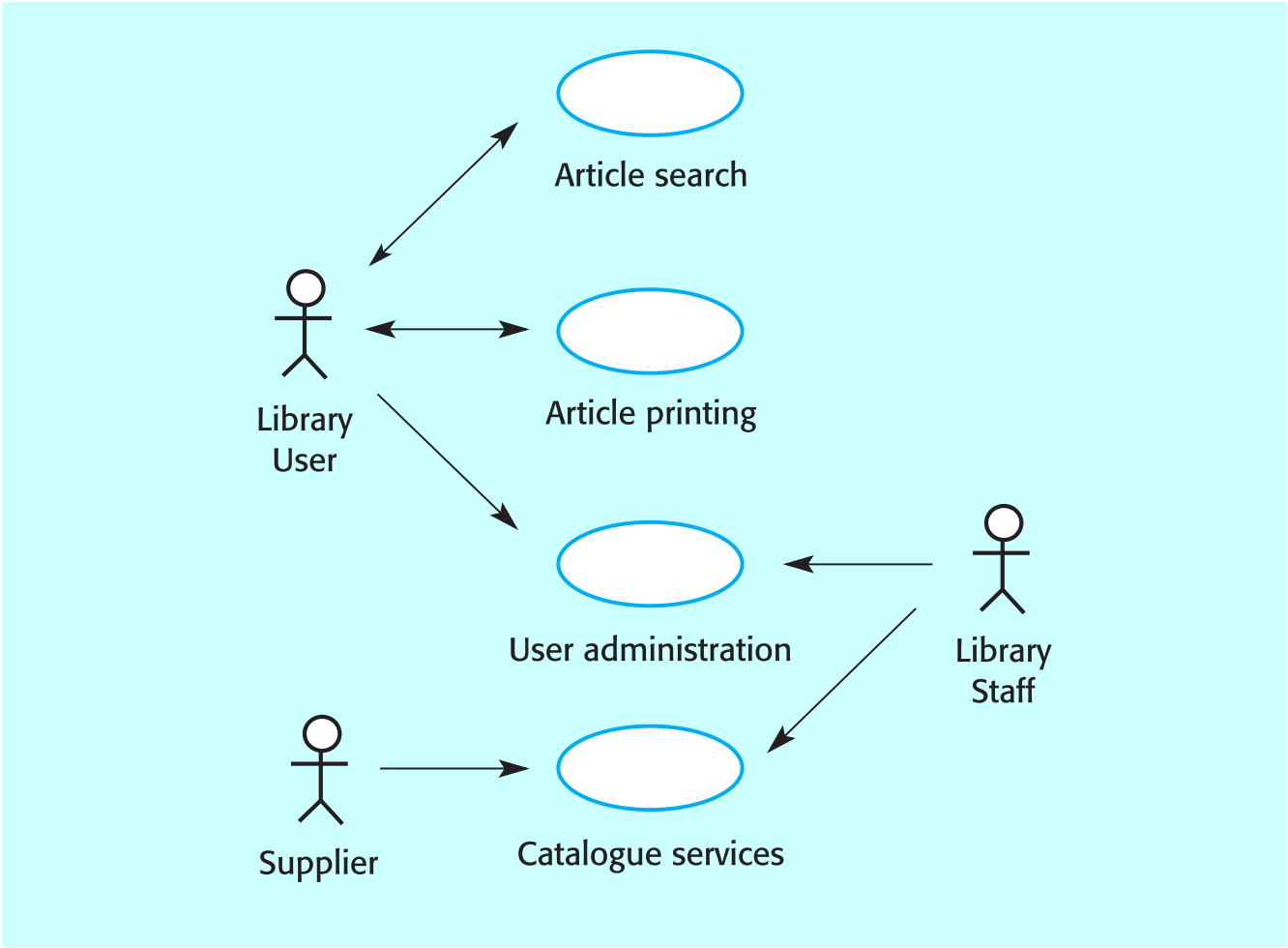
Scenario-Based Elicitation

- Scenarios may be written as text, supplemented by diagrams, screen shots, and so on
  - A more structured approach such as event scenarios or use-cases may be adopted

Use-Cases

- Use-cases are a scenario based technique which were first introduced in the Objectory method (Jacobson, 1993)
  - They have now become a fundamental feature of the UML
- Scenarios and use-cases are not effective for eliciting constraints or high-level business and non-functional requirements, or for discovering domain requirements
  - Because they focus on interactions

Use-Case Example



Ethnography

- Ethnography is an observational technique that can be used to understand social and organizational requirements
  - An analyst immerses him or herself in the working environment where the system will be used
  - He or she observes the day-to-day work
  - The ethnography helps analysts discover implicit system requirements that reflect the actual rather than the formal processes in which people are involved

Ethnography

- People often find it very difficult to articulate details of their work because it is second nature to them
  - Some factors that affect the work but that are not obvious to individuals may only become clear when noticed by an unbiased observer
- Ethnographic studies can reveal critical process details that are often missed by other requirements elicitation techniques
  - However, ethnography is not a complete approach and should be used to complement other approaches, such as use-case analysis

Apprenticing

- The requirements engineer actually conducts the tasks that users of the system will be carrying out
  - This has the added benefit that the requirements analyst gains critical insight into the tasks required of the new system

Questionnaires

- Questionnaires can gather large amount of data quickly, can be precisely targeted and are relatively inexpensive to run

- Problems with questionnaires
  - Designing questionnaires can be difficult, questions can be ambiguous and the lack of direct contact prevents clarification of issues
  - A poor response rate
  - The data can remain untested unless good follow-up methods are employed
  - A danger that the respondents may be careless in filling in questionnaires

Brainstorming

- Brainstorming can be an effective way to generate lots of ideas in a short space of time
  - It works by having a number of people in a relaxed informal environment
  - The statement or problem for which ideas are sought is stated and then the people in the session simply shout out ideas until the time limit is reached
  - All of the ideas are captured and after the meeting these ideas are sifted to determine which idea(s) best solves the problem
  - There are rules to a brainstorming session, for example, no criticism of ideas is allowed and no discussion of ideas is allowed
- Brainstorming is good for creative input into a requirements session, perhaps when something new is being sought or there is a difficult problem to overcome
  - Brainstorming is not appropriate for any kind of analysis or for decision making

Document Inspection

- Inspecting documents is a good means of gathering requirements where existing systems have been formalized
  - Documents are a good source of information for checking facts from other sources as they do not rely on staff cooperation
- Informal systems and processes, or those systems and processes that are not documented are typically missed
  - Further, there is a problem with documents eventually becoming dated and so there is a need for the requirements analyst to ensure that documents are current and "*live*"*!*
- Document inspection can be time consuming so there is often a need to be selective
  - Also, data may not be in an immediately usable form and there is usually no information on attitudes or opinions

Topics covered

- Requirement Engineering
- Requirements Elicitation
- Requirements Validation
- Requirements Management

Requirements Validation

- Requirements validation is concerned with showing that the requirements actually define the system that the customer wants
  - Requirements validation overlaps analysis in that it is concerned with finding problems with the requirements
  - Requirements validation is important because errors in a requirements document can lead to extensive rework costs when they are discovered during development or after the system is in service

Requirements Checking

- **During the requirements validation process, checks should be carried out on the requirements**

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

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
  - If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems
  - If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered
  - Developing tests from the user requirements before any code is written is an integral part of extreme programming

Topics covered

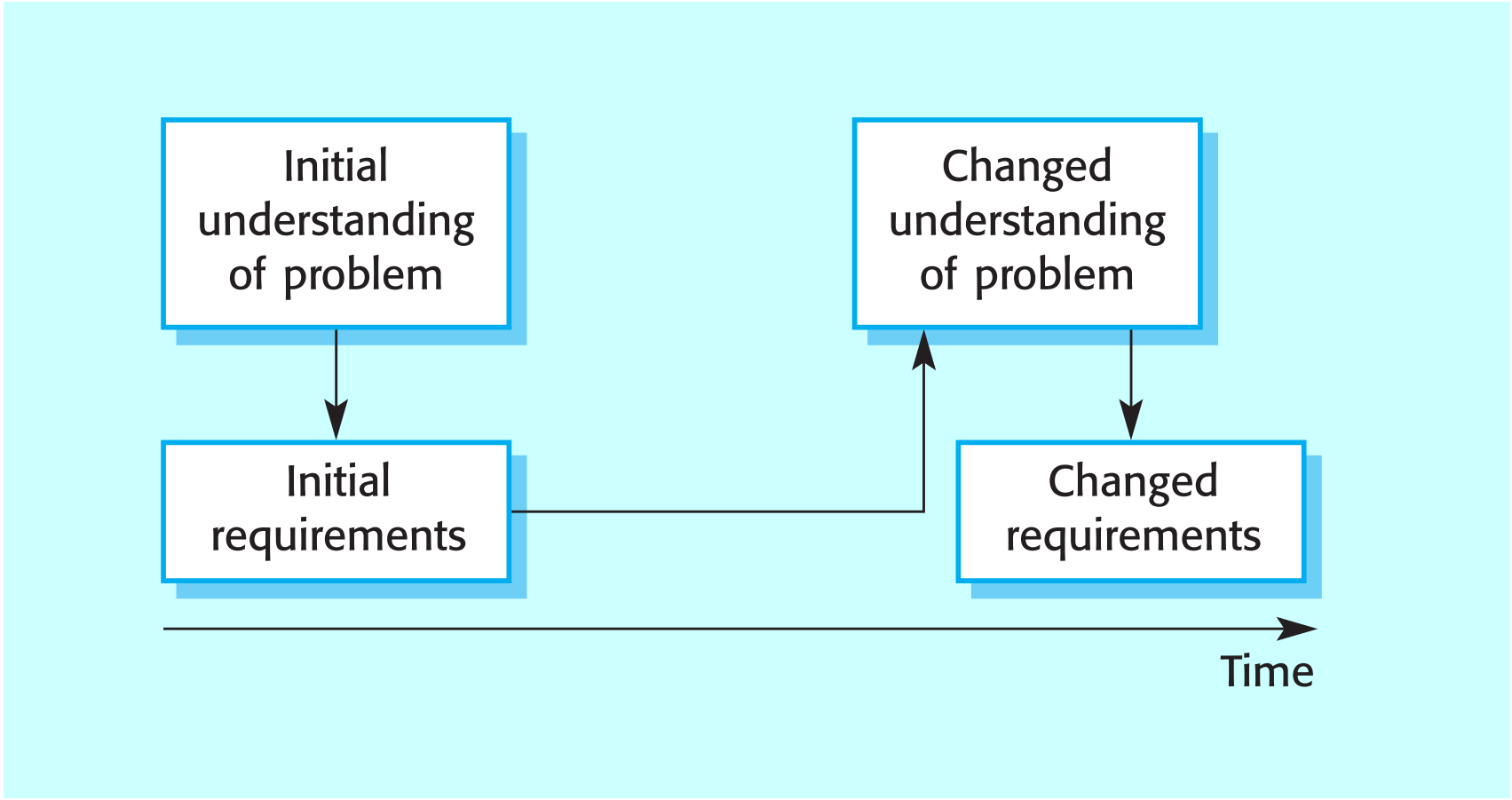
- Requirement Engineering
- Requirements Elicitation
- Requirements Validation
- Requirements Management

Requirements Management

- The requirements for large systems are always changing
  - Because the problem cannot be fully defined, the software requirements are bound to be incomplete
  - The stakeholders' understanding of the problem is constantly changing
  - Once a system has been installed, new requirements inevitably emerge

- Requirements management is the process of understanding and controlling changes to system requirements
  - You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes

Requirements Evolution



Enduring and Volatile Requirements

- **From an evolution perspective, requirements fall into two classes**

- **Enduring requirements**
  - **Relatively stable requirements derived from the core activity of the customer organisation**
    - E.g. A hospital will always have doctors, nurses, etc.
    - May be derived from domain models
- **Volatile requirements**
  - **Requirements which change during development or when the system is in use**
    - E.g. In a hospital, requirements derived from health-care policy

Traceability

- **Traceability is the property of a requirements specification that reflects the ease of finding related requirements**
  - **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;

Traceability matrices

- **Traceability information is often represented using traceability matrices**
  - **Each requirement is entered in a row and in a column in the matrix**
  - **A ‘D’ in the row/column intersection illustrates that the requirement in the row depends on the requirement named in the column**
  - **An ‘R’ means that there is some other, weaker relationship between the requirements**
    - For example, they may both define the requirements for parts of the same subsystem

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	