Software Development Life Cycle

Lecture 5: Requirements

DR. YOUNG SAENG PARK

2021/2022

WMG
THE UNIVERSITY OF WARWICK

Content

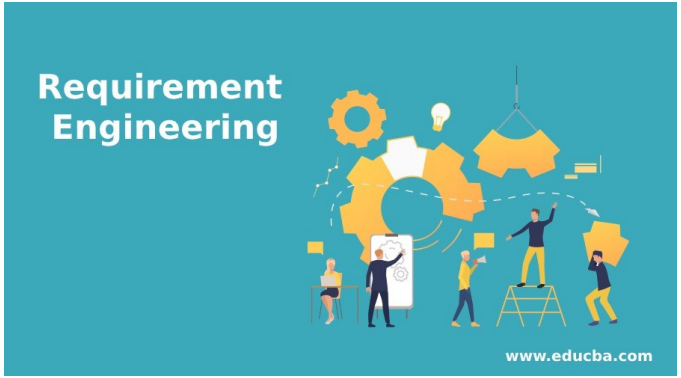
- Requirements Engineering
- Functional and Non-Functional Requirements
- Requirements Elicitation
- Requirements Specification
- Requirements Validation
- Requirements Change

Requirements

Requirements Engineering

Requirements Engineering

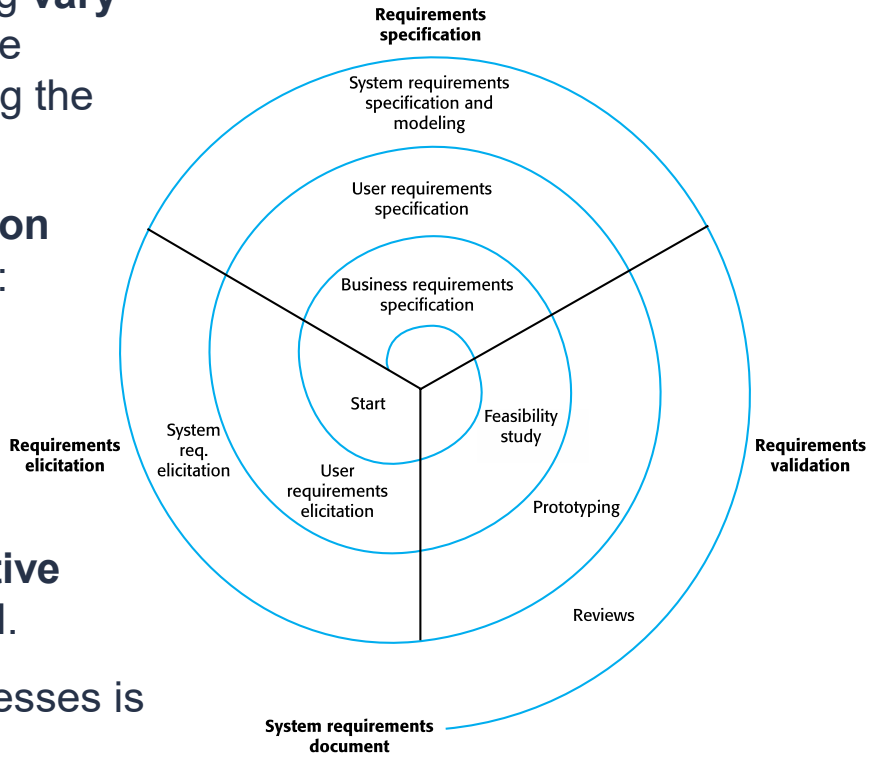
- **Requirements engineering (RE)** is the process of **defining**, **documenting**, and **maintaining** requirements in the engineering design process.
- It is a common role in **systems engineering** and **software engineering**.
- Typically, requirements engineering is presented **as the first phase of the development process**.
- **The activities** involved in requirements engineering **vary widely**, depending on the **type of system** being developed, and the **organisation's specific practices**.
- Requirements engineering has been shown to **clearly contribute to software project successes**.



Source: eduCBA

Requirements Engineering Processes

- The processes used for requirements engineering **vary widely** depending on the **application** domain, the **people** involved and the **organisation** developing the requirements.
- There are a number of **generic activities common processes** involved in requirements engineering:
  - **Requirements Elicitation**
  - **Requirements Specification**
  - **Requirements Validation**
- In practice, requirements engineering is **an iterative process in which the activities are interleaved**.
- The output of the requirements engineering processes is **a system requirements document**.



What is Requirements?

- The **requirements** for a system are **the description of the services** that a system should provide and **the constraints on its operation**.
- These requirements **reflect the needs of customers** for a system that **serves a certain purpose** such as making orders, booking reservations, searching information, etc.
- It may be **high-level abstract statements** or detailed **mathematical functional specifications** for a system.
- This is inevitable as requirements may serve a **dual function**:
  - May be the basis of a bid for a contract – therefore must be **easy to interpretation**.
  - May be the basis for the contract itself – therefore must be **defined in detail**.

Source: Shutterstock

Types of Requirement

User Requirements

- User requirements mean the **high-level abstract requirements**.
- The user requirements are statements in **natural languages** plus **diagrams** that the system is expected to provide users on its operational constraints.
- It should be **written for customers**.

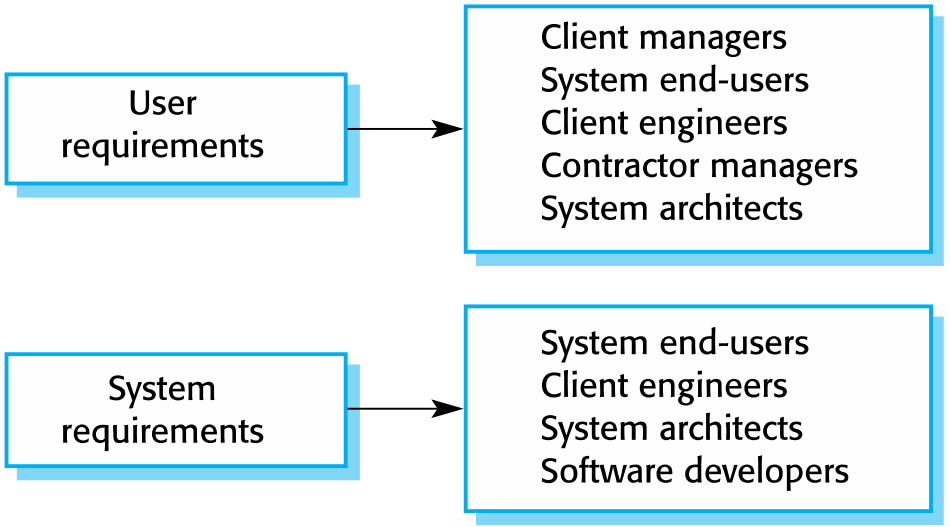
System Requirements

- System requirements **mean the detailed description** of what the system should do.
- The system requirements should **define exactly what is to be implemented**, including the system's **functions**, **services** and **operational constraints**.
- It should be **written for developers**.
- The system requirements document may be **part of a contract** between client and contractor.

Source: YouTube-Udacity

Different Readers of Requirements

- Different kinds of requirements are needed to **communicate** information about a system to **different types of reader**.
- **The readers of the user requirements** are usually concerned with **what the system will do.**
- **The readers of the system requirements** need to know more precisely **how the system will be implemented**.
- System stakeholders **include anyone who is affected by the system** in some ways.
- Thus, **the stakeholders range from** end-users of a system through managers to external stakeholders such as regulators.



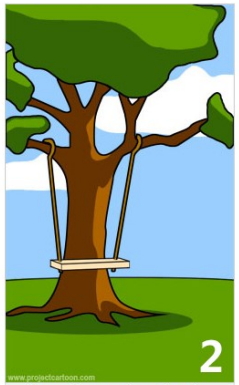
Requirements Imprecision

- Problems arise **when requirements are not precisely stated**.
- Changes in the requirements may **delay system delivery** and **increases costs**.
- **Imprecision in the requirements** can lead to **disputes between users and software developers**.
- **Ambiguous requirements** may **be interpreted in different ways** by developers and users.
- Thus, it is very important to constantly check any gap between **user intention** and **developer interpretation** in order to avoid requirements imprecision.

Source: Medium

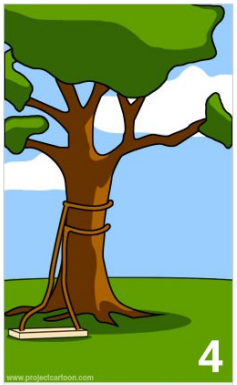
Requirements Imprecision Example

How the customer explained it

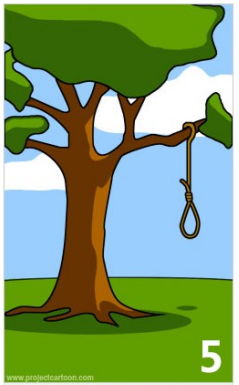


How the project leader understood it

How the analyst designed it

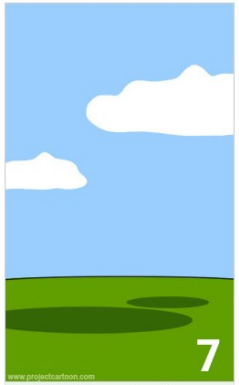


How the programmer wrote it

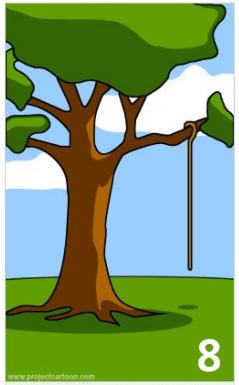


What the beta testers received

How the business consultant described it

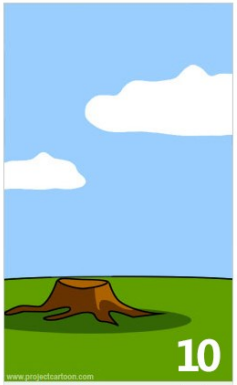


How the project was documented



What operations installed

How the customer was billed



How it was supported

What marketing advertised

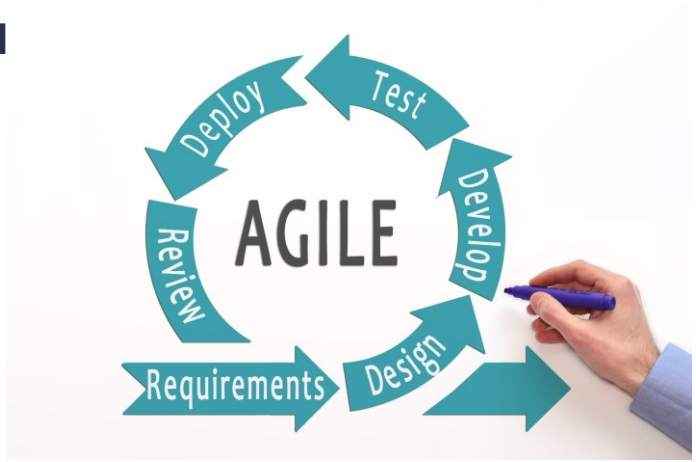
What the customer really needed

Requirements Completeness and Consistency

- In principle, requirements should be **both complete and consistent**.
- **Completeness**
  - They should include descriptions of all services and information required by users.
- **Consistency**
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- However, because of system and environmental complexity, it is **impossible to produce a complete and consistent** requirements document for large, complex systems.
- In practice, it may be only possible achieve requirements **consistency and completeness for very small software systems**.

Requirements – Agile Development

- Many agile development argues that **producing detailed system requirements is a waste of time** because requirements **change so quickly**.
- In traditional requirements development such as waterfall, the requirements document is **often out of date**.
- Agile development usually uses **incremental and iterative requirements engineering**.
- Agile development may express requirements **as 'user stories'** rather than functional specification.
- The requirements collecting using agile development is **practical for business systems** but **problematic for systems that requirement pre-delivery analysis** (e.g. critical systems) or systems developed by several teams.



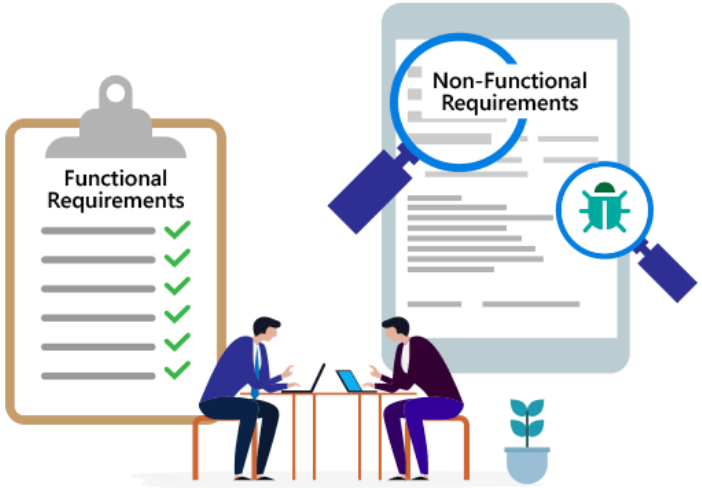
Source: Obeya Association

Requirements

Functional and Non-Functional Requirements

Functional and Non-Functional Requirements

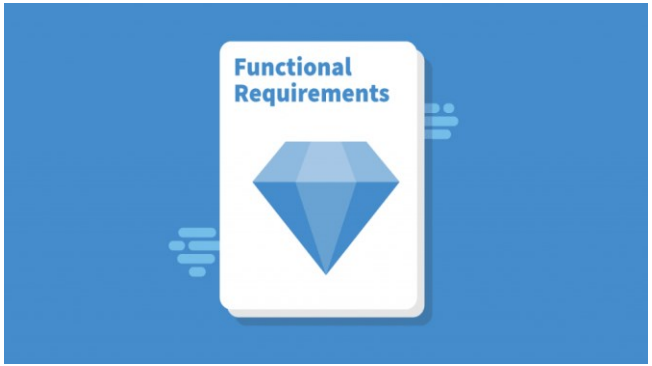
- Software system requirements are often **classified as functional** or **non-functional requirements**.
- **Functional Requirements**
  - **Statements of services** the system should provide, **how the system should react** to particular inputs and **how the system should behave** in particular situations.
  - May explicitly state what the system should not do.
- **Non-Functional Requirements**
  - **Constraints on the services or functions** offered by the system such as timing constraints, constraints on the development process, constraints imposed by standards, etc.
  - Often **apply to the system as a whole** rather than individual features or services.



Source: Morden Requirements

Functional Requirements

- Functional requirements describe **functionality** or **system services**.
- Functional requirements **expand the user requirements** and are **written for system developers**.
- Functional requirements should **describe the system services in detail** such system functions, their inputs and outputs, exceptions, etc.
- Functional requirements **vary from general requirements** covering what the system do **to very specific requirements** reflecting an organisation's needs.
- Functional requirements **may involve calculations**, **technical details**, **data manipulation and processing, and other specific functionality** that define what a system is supposed to accomplish.



Source: DevTeam.Space

Non-Functional Requirements

- Non-functional requirements are **requirements that are not directly concerned with the specific services** delivered by the system to its users.
- Non-functional requirements may be **more critical than individual functional requirements**.
- These define **system properties and constraints** e.g. **reliability**, **response time** and **storage requirements**.
- Failing to meet a non-functional requirements can mean that **whole system is unusable**.
- For example, if an aircraft system does **not meet its reliability requirements**, it will **not be certified** as safe for operation.
- For example, if an embedded control system **fails to meet its performance requirements**, the control functions will **not operate correctly**.

Source: ArtOfTesting

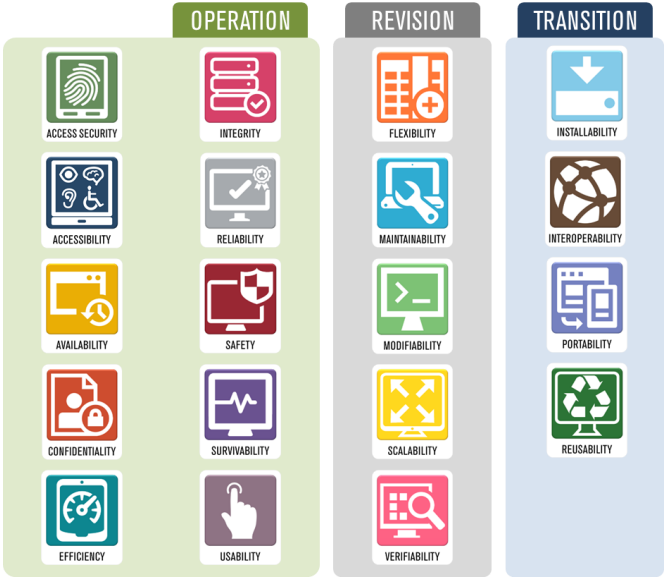
Non-Functional Requirements Implementation

- 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 organise the system to minimise 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 required.
  - For example, it may limit access to information in the system.
- Thus, the implementation of non-functional requirements **may be spread throughout the system**.
- Also, the implementation of non-functional requirements is often **more difficult** than functional requirements.

Source: Implementation Engineers

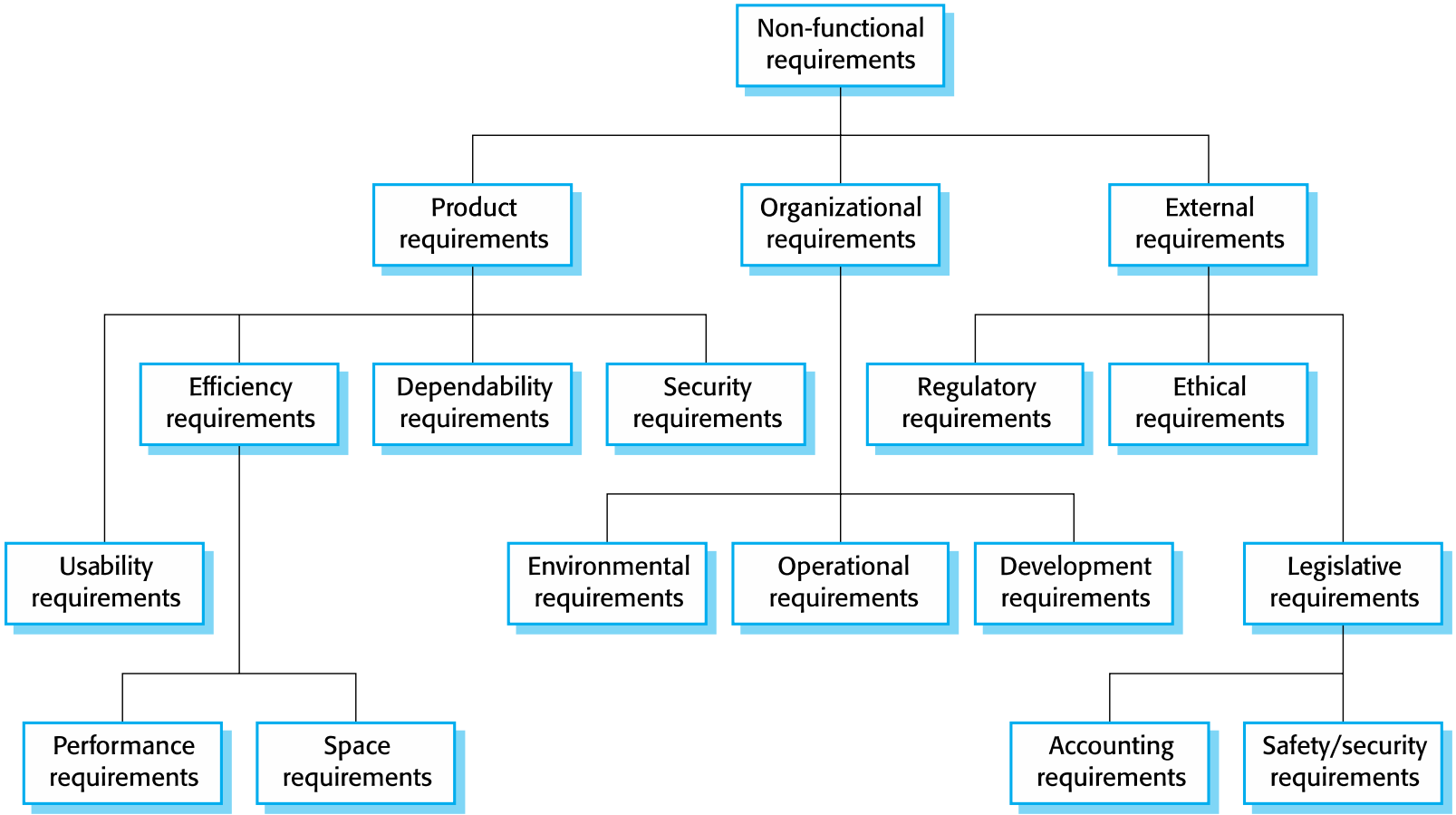
Non-Functional Classifications

- Non-functional requirements may come from typically product requirements, organisational requirements, or external requirements.
- **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.



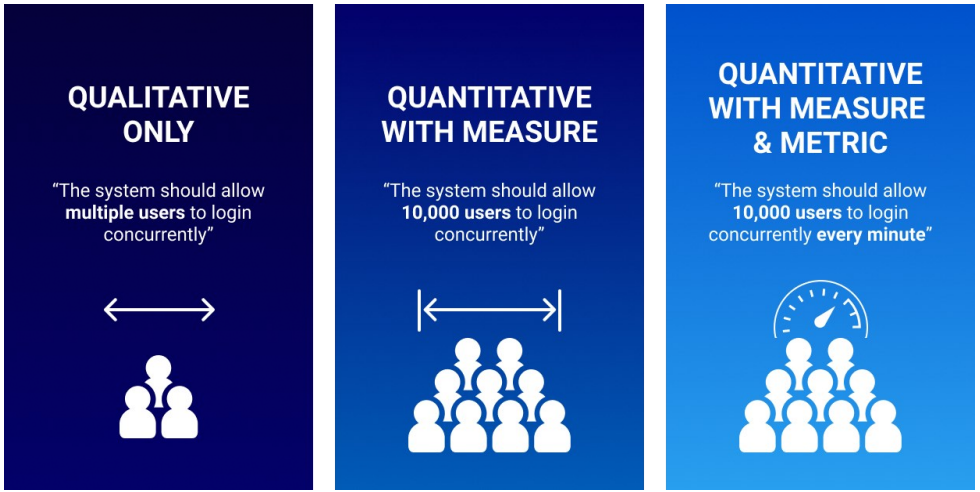
Source: Requirements Quest

Non-Functional Classifications



Quantitative Non-functional Requirements

- Non-functional requirements may be **very difficult to state precisely** and imprecise requirements may be difficult to verify.
- For example, **stakeholders propose requirements as general goals** such as ease of use, ability of the system to recover from failure, or rapid user response.
- Whenever possible, non-functional requirements should be **written in quantitative form** such as a statement using some measure that can be objectively tested.



Source: Jama Software

Metrics for Specifying Non-Functional 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 unavailable 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

Requirements Elicitation

Requirements Elicitation

- The aim of the requirements elicitation process is to **understand the work that stakeholders do** and **how they might use a new system** to help support that work.
- During requirements elicitation, **software engineers work with stakeholders** to find out the followings:
  - Application domain
  - Work activities
  - Services and system features
  - System's operational constraints
  - Hardware constraints
  - Required performance of the system

...

Source: LnkedIn

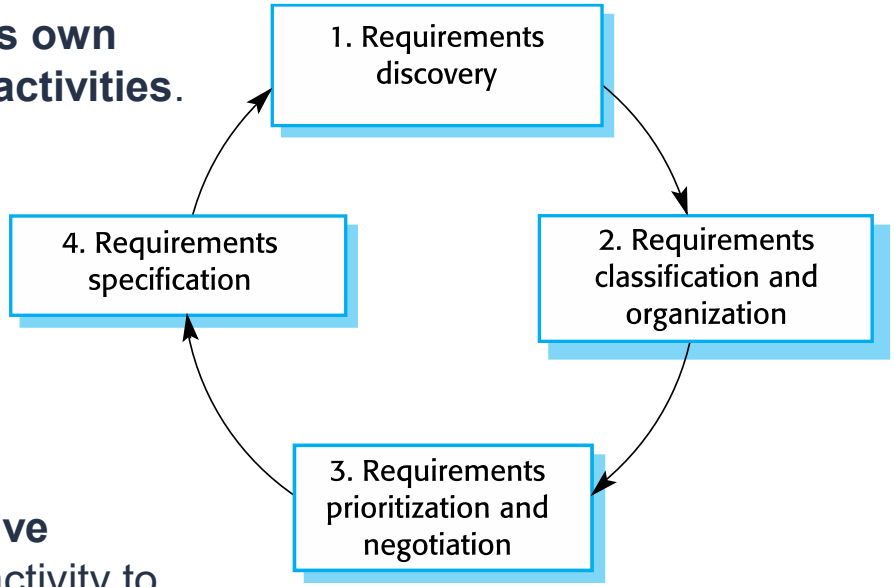
Problem of Requirements Elicitation

- **Eliciting** and **understanding** requirements from stakeholders **is difficult process for several reasons**:
  - Stakeholders often **don't know what they really want** from a computer system.
  - They may make **unrealistic demands** because they don't know **what is and isn't feasible**.
  - Stakeholders **express requirements in their own terms**.
  - **Different stakeholders with diverse requirements** may have conflicting requirements.
  - **Organisational and political factors** may influence the system requirements.
  - **The requirements change** during the analysis process due to the economic and business environment.
  - **New stakeholders** may emerge and **the business environment** may change.

Source: YouTube-Udacity

Requirements Elicitation Process

- **Depending on local factors** such as the expertise of the staffs, the type of system being developer, and standard used, **each organisation will have its own version of requirements elicitation process activities**.
- The typical process activities include:
  - Requirements Discovery
  - Requirements Classification and Organisation
  - Requirements Prioritisation and Negotiation
  - Requirements Specification
- The requirement elicitation process is **an iterative process with continual feedback** from each activity to other activities.
- The cycle ends when **the requirements document** has been produced.



Requirements Elicitation Process Activities

Requirements Discovery

- This is the process of interacting with stakeholders to discover their requirements.
- Domain requirements from stakeholder and documentation are discovered at this stage.

Requirements Classification and Organisation

- This activity takes the unstructured collection of requirements.
- It groups related requirements and organises them into coherent cluster.

Requirements Prioritisation and Negotiation

- This activity is concerned with prioritising requirements, and finding and resolving requirements conflicts through negotiation
- Usually, stakeholders have to mee to resolve differences and agree on compromise requirements.

Requirements Specification

- The requirements are documented and input into the next round of the spiral.
- An early draft of the software requirement documents may be produced at this stage.

Requirements Elicitation Techniques

- Requirements elicitation involves meeting with stakeholders **to discover information about the proposed system** such as**:**
  - How people work.
  - What they produce.
  - How they use other systems.
  - How they may need to change to accommodate a new system.

  ...
- There are two fundamental approaches to requirements elicitation:
  - **Interviewing**, where you talk to people about what they do.
  - **Observation or Ethnography**, where you watch people doing their job, what artifacts they use, how they use them, and so on.
- It is better to use **a mix of interviewing and observation** to collect information.

Source: Humans of Data - Atlan

Requirements Elicitation Techniques – Interviewing

- Interviews are **good for getting an overall understanding** of such as what stakeholders do, how they might interact with the new system and the difficulties that they face.
- **Questions or agenda** are given to stakeholders and the requirements are derived from the stakeholders' answers.
- Interviews may be of two types:
  - **Closed interviews** based on **pre-determined list of questions**.
  - **Open interviews** where various issues are explored with stakeholders **without predefined agenda**.
- There are problems with Interviews:
  - Application specialists may use specific language to describe their area of work. So, it may be **difficult to discuss and understand requirements** without the specific language.
  - Some domain knowledge is so familiar to stakeholders but they find it **difficult to explain**.

Source: ClearVoice

Requirements Elicitation Techniques – Interviewing

- To be an effective interviewer, you should:
  - Be open-minded.
  - Use appropriate language.
  - Avoid pre-conceived ideas about the requirements.
  - Be willing to listen to stakeholders.
  - Be willing to change your mind about the system.
  - Prompt the interviewee to get discussions going using a requirements proposal, or by working together on a prototype system.
  - Talk in a defined context rather than in general terms.
  - Practice note-taking
  - Focus on the conversation.
  - Choose questions carefully.

...

Source: ClearVoice

Requirements Elicitation Techniques - Ethnography

- People **understand their work** but **may not understand its relationship to other work** in the organisation.
- Social and organisational factors which are not obvious to individuals may be **observed by an unbiased observer**.
- **Ethnography** is an **observational technique** that can be used to understand operational processes and help to discover implicit system requirements.
- Ethnography is particularly effective for discovering two types of requirements:
  - Requirements derived from the way in which **people actually work** rather than the way in which business process definitions suggest.
  - Requirements derived from **cooperation and awareness of other people's activities**.
- Ethnography is **effective for understanding existing processes** but this understanding does not **always help with innovation**. For example, **Nokia** used ethnography to discover how people used their phone and developed new phone models based on the discovers. **Apple** ignored current use and revolutionised the mobile phone industry, **iPhone**.

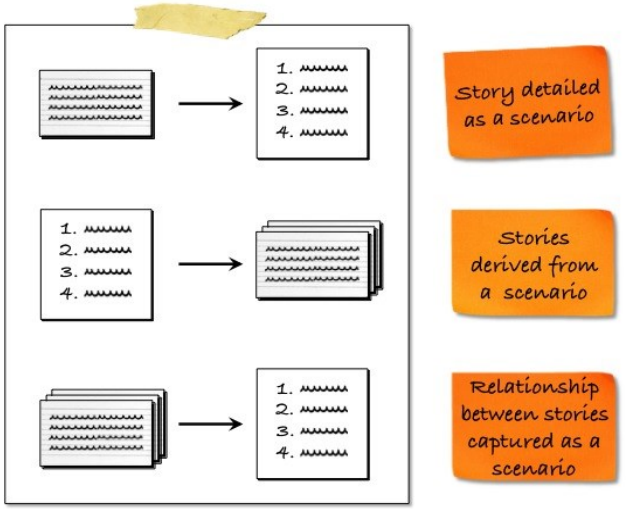
Requirements Elicitation with Stories and Scenarios

- People find it **easier to relate to real-life examples** than abstract descriptions.
- Stories and scenarios are **ways of capturing information** from stakeholders.
- Stories and scenarios can be used to **develop more specific system requirements** during interviewing groups of stakeholders to discuss the system.
- The difference between stories and scenarios is **the level of detail structure**.
  - Stories are **written as narrative text** and present a high-level description of system use.
  - Scenarios **are usually structured** with specific information collected such as inputs and outputs.

Source: Designmodo

Requirements Elicitation with Stories and Scenarios

- **The advantage of stories** is that everyone can **easily related to stakeholders**.
- **A interview with stories is useful** to get information than a interview without it.
- **The high-level stories** do not go into detail about a system but they **can be developed into more specific scenarios**.
- Thus, a scenario is **a structured form of user stories** and a scenario includes such as:
  - A description of **what the system and users expect** when the scenario starts.
  - A description of **the normal flow of events** in the scenario.
  - A description of **how the resulting problems can be handled**.
  - A description of **other concurrent activities**' information.
  - A description of **the system state** when the scenario finished.



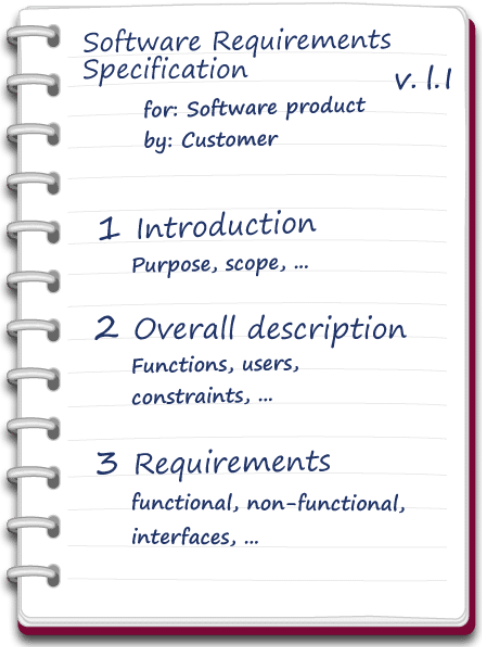
Source: Roman Pichler

Requirements

Requirements Specification

Requirements Specification

- Requirement specification is **the process of writing down the user and system requirements** in a requirements document.
- User requirements **are generally written in natural language** with appropriate **diagrams** and **tables** in the requirements documents.
- When writing user requirements, it **should not use software jargon, structured notation or formal notions**.
- System requirements **is also written in natural languages but other notations** based on forms, graphical or mathematical system models.
- When writing system requirements, it should **add detail and explain how the system should provide** the user requirements.



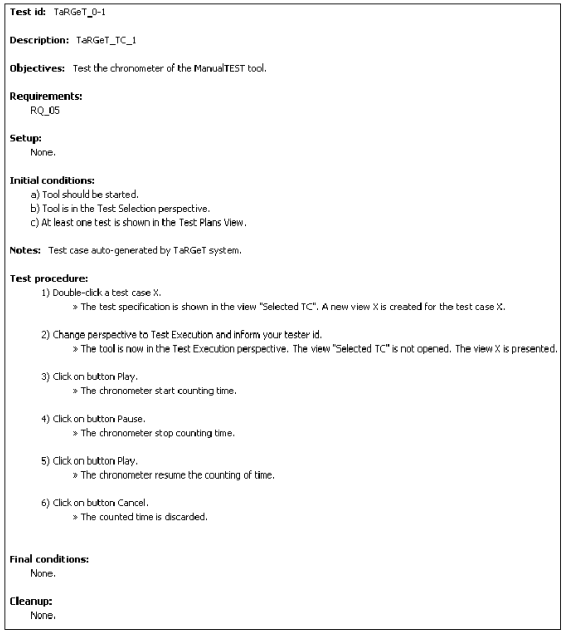
Source: microTOOL

Requirements Specification Notation for Writing

Notation	Description
Natural Language Sentences	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 Language	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 model, 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**.

Natural Language Specification

- Natural language has been used to write requirements for software **since the 1950s**.
- It is **expressive**, **intuitive**, and **universal**. However, its interpretation **may be ambiguous**, depending on the background of the reader.
- To minimise misunderstandings when writing natural language requirements:
  - **Invent a standard format** and ensure that all requirement definitions are included in that format.
  - **Use languages consistently** to distinguish between mandatory and desirable requirements.
  - **Use text highlighting** (bold, italic, or colour) to pick out key parts of the requirements
  - **Do not assume that readers understand** technical, software engineering language.



Test id: TaRGeT_0-1

Description: TaRGeT_TC_1

Objectives: Test the chronometer of the ManualTEST tool.

Requirements:
RQ_05

Setup:
None.

Initial conditions:
a) Tool should be started.
b) Tool is in the Test Selection perspective.
c) At least one test is shown in the Test Plans View.

Notes: Test case auto-generated by TaRGeT system.

Test procedure:
1) Double-click a test case X.
» The test specification is shown in the view "Selected TC". A new view X is created for the test case X.

2) Change perspective to Test Execution and inform your tester id.
» The tool is now in the Test Execution perspective. The view "Selected TC" is not opened. The view X is presented.

3) Click on button Play.
» The chronometer start counting time.

4) Click on button Pause.
» The chronometer stop counting time.

5) Click on button Play.
» The chronometer resume the counting of time.

6) Click on button Cancel.
» The counted time is discarded.

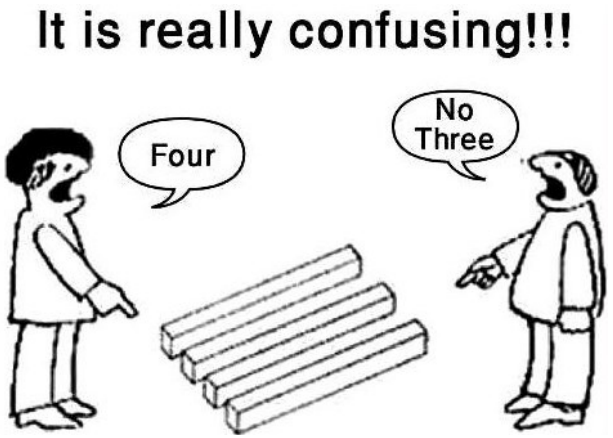
Final conditions:
None.

Cleanup:
None.

Source: ResearchGate

Natural Language Specification Problem

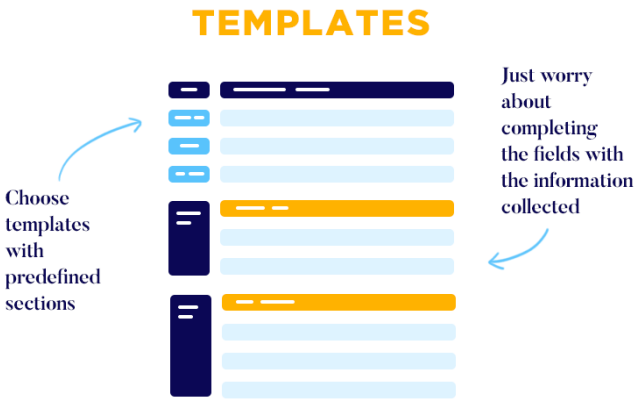
- Various problems are arise when requirements are written in natural language sentences.
- **Lack of Clarity**
  - It is difficult to use language in a precise and unambiguous way without making the document wordy and so difficult to read.
- **Requirements Confusion**
  - Functional and non-functional requirements tend to be mixed-up and so may not be clearly distinguished.
- **Requirements Mixture**
  - Several different requirements may be expressed together as a single requirement.
- **Over-flexibility**
  - The same thing may be described in different ways in the specification.



Source: cumedyhut

Structured Specifications

- Structured natural language is **a way of writing system requirements where requires are written in a standard way** rather than as free-form text.
- This approach ensures that **some uniformity is imposed** on the specification.
- Structured language notations **use templates to specify system requirements**.
- Typically, whenever possible, **the templates can be reused** from other specifications.
- The specification may be structured around the followings:
  - **The objects** manipulated by the system
  - **The functions** performed by the system
  - **The events** processed by the system.



Source: Justinmind

Structured Specifications with Form

- When a standard format is used for specifying functional requirements, the following information should be included:
  - A description of **the function or entry** being specified.
  - A description of **its inputs and the origin of these inputs**.
  - A description of **its outputs and the destination of these outputs**.
  - **Information needed for the computation and other entities** in the system.
  - A description of **the action to be taken**.
  - A description of **pre and post conditions** (if appropriate) which se out what must be true before/after a function is called.
  - A description of **the side effects** (if any) of the function.

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.

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.

Requirements

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.

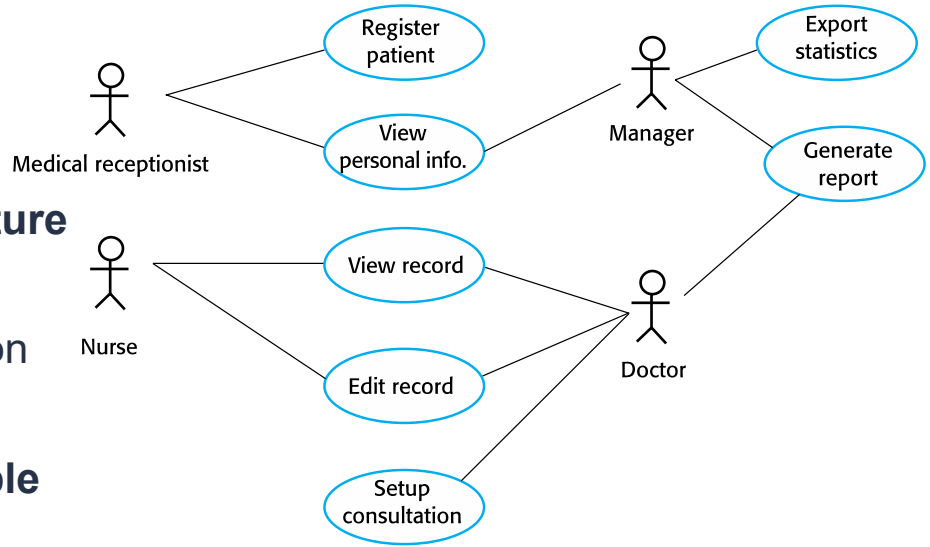
Structured Specifications with Tables

- However, structured specification is still sometimes **difficult to write requirements in clear and unambiguous way**, particularly when **complex computations** are to be specified.
- To address this problem, it might be better to add extra information using other formats such as **tables** or **graphical model of the system**.
  - How computation proceed
  - How the system state changes
  - How users interact with the system
  - How sequences of actions are performed.
- **Tables** are particularly useful when there are **a number of possible alternative situations**.

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 & ((r2 − r1) ≥ (r1 − r0))	CompDose = round ((r2 − r1)/4) If rounded result = 0 then CompDose = MinimumDose

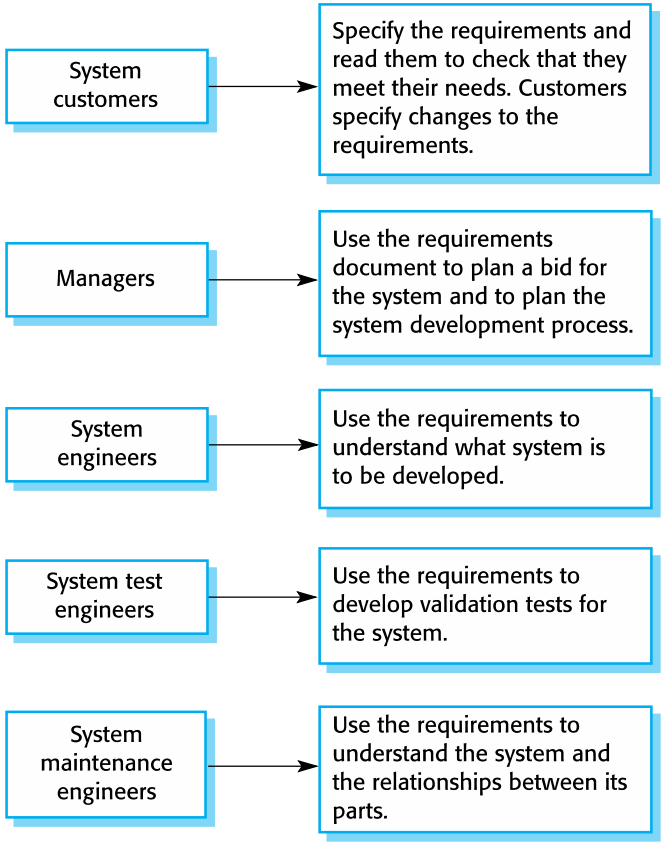
Structured Specifications with Graphical Model

- **Use cases** are **a way of describing interactions between users and a system** using graphical model and structured text.
- **Use cases** have become a **fundamental feature of Unified Modelling Language (UML)**.
- 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**.
- Use cases are documented using **a high-level use case diagram**.
- **The additional information** may be a **textual description** or one or **more graphical models** such as **UML sequence** or **state charts**.



Software Requirements Document

- **The software requirements document** (also called the software requirements specification) is **an official statement** of what the developers should implement.
- The document should **include** both **the user requirements** for a system and detailed specification of **the system requirements**.
- However, agile methods argue that **requirements change so rapidly** that a requirements document is often **out of date** and the effort **is largely wasted**.
- Typically, **the user requirements** are described in an **introductory chapter** in the system requirements specification.
- The requirements document has **a diverse set of user**, ranging from **the senior management** of the organisation to **the engineers** responsible for developing the software.



Software Requirements Document Variability

- **The level of detail information** in a requirements document **depends on type of system** being developed and **the development process** used.
- For example, **critical systems** need **detailed requirements** because safety and security have to be analysed in detail.
- For example, if **an in-house system** is developed, iterative development process is used and the requirements document can **be less detailed**.
- One of requirements documents standard have been designed by **the U.S Institute of Electrical and Electronic Engineers (IEEE)**, e.g. IEEE standard (IEEE 1998).
- The IEEE standard is a **generic one that can be applicable** to the requirements for large systems engineering projects.

IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Recommended Practice for Software Requirements Specifications

Sponsor
Software Engineering Standards Committee of the IEEE Computer Society

Approved 25 June 1998
IEEE-SA Standards Board

Abstract: The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.
Keywords: contract, customer, prototyping, software requirements specification, supplier, system requirements specifications

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Print: ISBN 0-7381-0332-2, SH94654
PDF: ISBN 0-7381-0448-5, SS94654

Source: IEEE

IEEE Standard of Requirements Document (1)

Notation	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.
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	This should describe **the services provided for the user**. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customer.
System Architecture	This chapter should present a **high-level overall of the anticipated system architecture**, showing the distribution of functions across system modules.

IEEE Standard of Requirements Document (2)

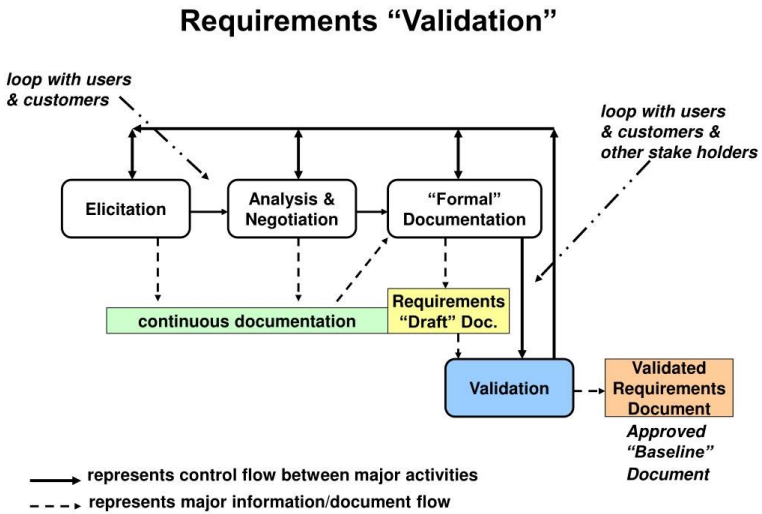
Notation	Description
System Requirements	This should describe **the functional and non-functional requirements in more detail**. If necessary, further detail may also be added to the non-functional requirements.
System Models	This might include **graphical system models showing the relationships between 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 systems.
Appendices.	These should provide **detailed, specific information** that is related to the application being developed; for example, hardware and database descriptions.
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, etc.

Requirements

Requirements Validation

Requirements Validation

- Requirements validation is **the process of checking that requirements define the system** that customers want.
- Requirements validation is **critically important** because errors in a requirements document **can lead to extensive rework cost**.
- **Fixing a requirement error after delivery may cost up** to 100 times the cost of fixing an implementation error.
- **A change to the requirements** usually means that **the system design and implementation** must also **changed**.
- Furthermore, **the system must be retested**.



Source: SlideServe

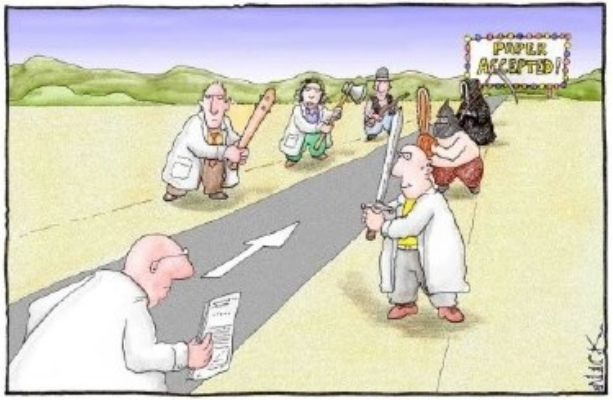
Requirements Checking

- During the requirements validation process, **different type of checks should be carried out** on the requirements:
- **Validity Checks**
  - Does the system provide the functions which support the customer's need?
- **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 Checks**
  - Can the requirements be verifiable?

Source: Project Community

Requirements Validation Techniques

- A number of requirements validation techniques can be used **individually or in conjunction with one another**.
- **Requirements Reviews**
  - The requirements are analysed **systematically by a team or reviewers** who check for errors and inconsistencies.
- **Prototyping**
  - This involves **an executable model of a system** and checks requirements **by end-users and customers** to see whether it meets their needs and expectation.
- **Test-case Generation**
  - This approach involves **developing test-cases for requirements** and then check system's testability.
  - **Developing test-cases** from the user requirements before any code is written is **an integral part of test-driven development**.



Source: SlideShare

Requirements Reviews

- A requirements review is **a manual process** that involves people from **both client and contractor staff**. A requirements reviews can be **informal** or **formal**.
- A requirements review **may be organised** as a broader activity with different people **checking different part of the document**.
- Informal requirements review involves **discussing requirements with as many system stakeholders** as possible.
- Formal requirements reviews should **check with the client through the system requirements** explaining **the implications** of each requirement.
- The review team should **check each requirement for consistency** and should check **the requirements as a whole for completeness**. Reviews may also check for
  - **Verifiability**: Is the requirement realistically testable?
  - **Adaptability**: Can the requirement be changed without a large impact on other requirements?
  - **Comprehensibility**: Is the requirement properly understood?
  - **Traceability**: Is the origin of the requirement clearly stated?

Requirements

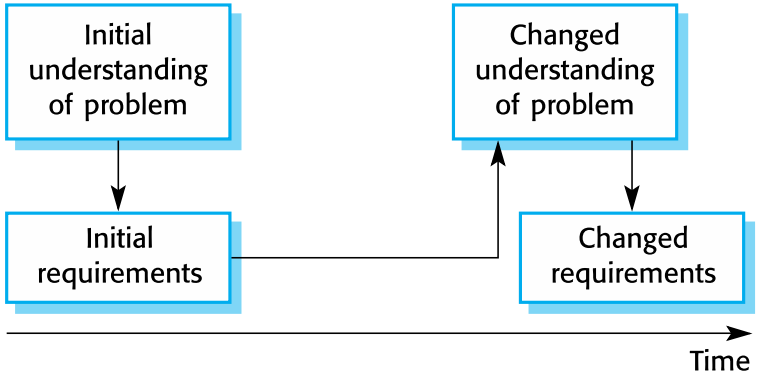
Requirements Change

Requirements Change in Business Environment

- The **business and technical environment** of the system **always changes** after installation.
  - For example, **new hardware may be introduced**; it may be necessary to interface the system with other systems; business priorities may change; **new regulations may be introduced** that the system must necessarily abide by.
- The **people** paying for a system and the **users** of that system are **rarely the same people**.
  - System customers impose requirements because of organisational and budgetary constraints. **These requirements 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.
- Large systems usually have **a diverse user community** with many users having **different requirements and priorities**.
  - The final system requirements are inevitably a compromise between them and it is often discovered that **the balance of support given to different users has to be changed and re-prioritised**.

Requirements Change Management

- As requirements are evolving, it is **necessary to establish a formal process for making change proposals** and linking these to system requirements.
- Requirements management is **the process of managing changing requirements** during the requirements engineering process and system development.
- **Agile development processes** have been designed to cope with requirements that **change during the development process**.
- However, in systems with multiple stakeholder, the requirements change will **benefit some stakeholders and not others**.
- It is necessary to **balance the needs of all stakeholders** and bring out the agreement.



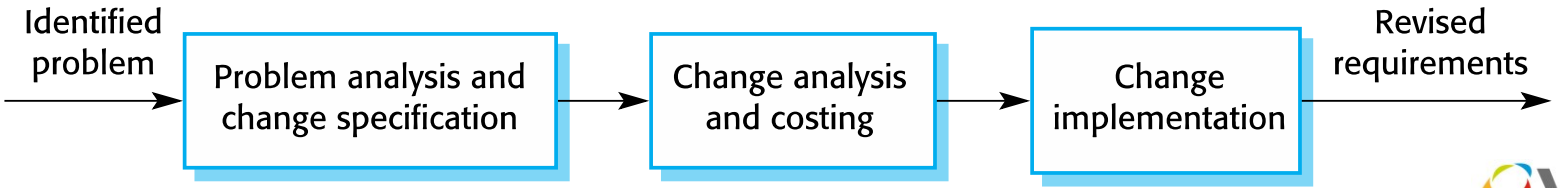
Requirements Change Management Planning

- Requirements management planning is to **establish how a set of evolving requirements** will be managed.
- Requirements management planning should consider:
  - **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 access the impact and cost of changes.
  - **Traceability policies**: These policies define the relationship 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.

Source: Oracle Blogs

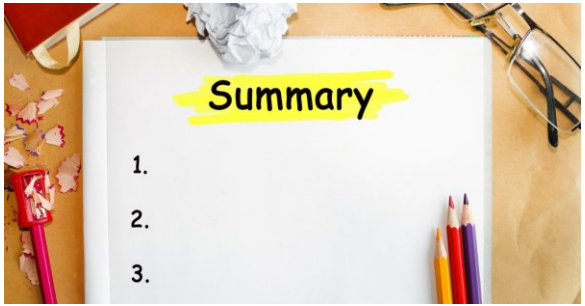
Requirements Change Management

- There are **three principal stages** in requirements change management:
  - **Problem analysis and change specification**: During this stage, the problem or the change proposal is analysed 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 accessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
  - **Change Implementation**: The requirements document and the system design and implementation are modified. Ideally, the document should be organised so that changes can be easily implemented.



Summary

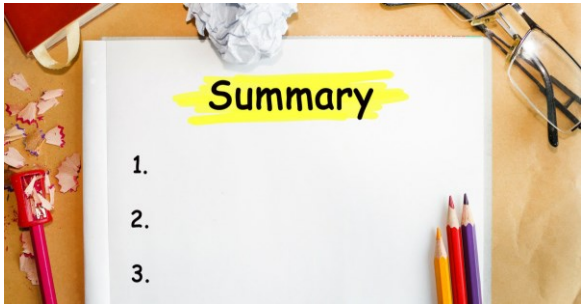
- **Requirements** for a software system set out **what the system should do** and define constraints on its operation and implementation.
- **Functional requirements** are statements of the services that the system must provide or are descriptions of **how some computations must be carried out**.
- **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**.
- The requirements engineering process includes **requirements elicitation**, **requirements specification**, and **requirement validation**.
- Requirements elicitation is the process to **understand the work that stakeholders do** and **how they might use a new system**.



Source: Elsevier

Summary

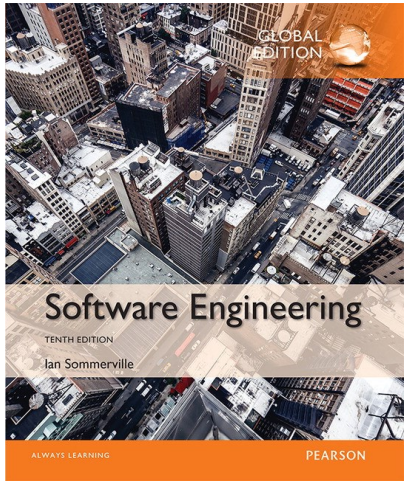
- Requirement specification **is the process of formally documenting the user and system requirements** and creating a **software requirements document**.
- The software requirements document is **an agreed statement of the system requirements.** It should be organised so that both **system customers and software developers** can use it.
- Requirements validation is **the process of checking the requirements** for validity, consistency, completeness, realism and verifiability.
- Business, organisational and technical changes **inevitably lead to changes to the requirements for a software system**. Requirements change management is the process of managing and controlling these changes.



Source: Elsevier

REFERENCE

[Main References]



Software Engineering
Ian Sommerville

Requirements engineering

[Other References]

- Wikipedia:
  - https://en.wikipedia.org/wiki/Requirements_engineering
  - https://en.wikipedia.org/wiki/Requirements_management
  - https://en.wikipedia.org/wiki/Software_requirements_specification
  - https://en.wikipedia.org/wiki/Requirements_elicitation
  - https://en.wikipedia.org/wiki/Change_management
  - https://en.wikipedia.org/wiki/Requirement
- QRA: https://qracorp.com/functional-vs-non-functional-requirements/
- PERFORCE: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document
- JavaTpoint: https://www.javatpoint.com/software-engineering-requirement-engineering
- OmarElgabry's Blog: https://medium.com/omarelgabrys-blog/requirements-engineering-introduction-part-1-6d49001526d3

THANKS!

Any questions?

y.s.park@warwick.ac.uk