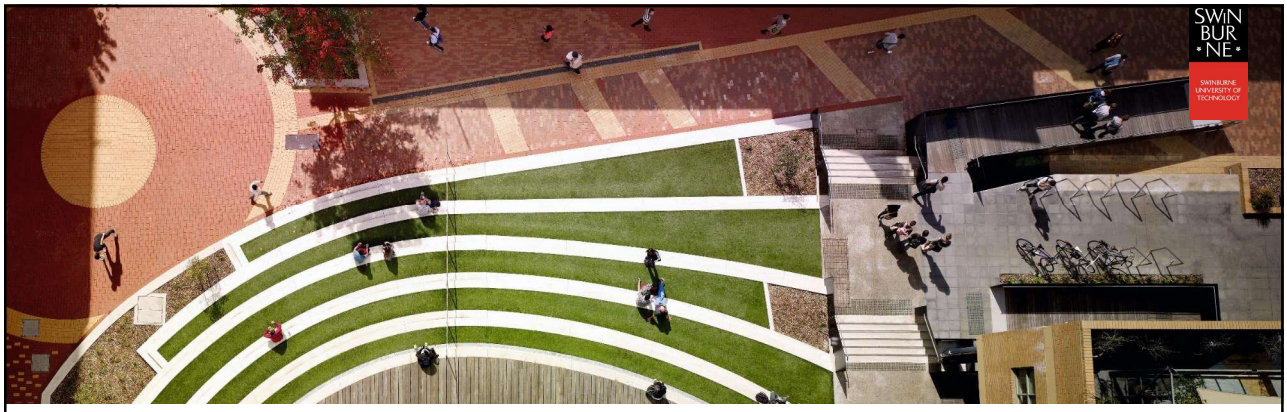


COS80022 – Software Quality and Testing

Week 2 – Software Requirements Specification

Outline

- Why is software built?
- Understanding Problems
- Functional and non-functional requirements
- Requirements document & specification
- Requirements engineering processes
- Requirements elicitation and analysis
- Requirements validation

Problem Domain

Why is software built?

- Software is often built to help business clients or users solve their problems (called "pain points")
  - What is not working properly?
  - What is hurting them?
  - How serious is the pain?
- Example
  - Taxis are so expensive & mostly unavailable in my area
- The clients/users can often express their pain points, but not the solution

Example: Business Systems

- Business goals with respect to
  - Supplier – reduce cost, improve predictability/quality
  - Customer – increase client base, keep existing clients
  - Employee – increase productivity, retain talent
- Business systems are developed to support these business activities
- Examples – MYOB
  - Improves *employee* productivity
  - Keeps *suppliers* happy (payments on time)
  - *Customer* retention (accurate and timely feedback to the customer)

Example: Software Systems

- SAP Supplier Management
- Oracle's Siebel CRM
- HRM System
- Recruitment Software System
- Inventory Management System
- Windows 10
- Excel
- Fortnite
- PES

What is the Problem?

- Software is built to address "pain/pleasure points"
- To ensure success, we need to pay proper attention to what is not working properly, what is hurting us
  - Pain/pleasure points
- Problem exists in a context
  - We need to understand the context to understand the "Pain Points"
- Once we identify the key high-level pain/pleasure points, further analysis will result in a detailed view of the problem

How much do you understand the problem?

- To better understand the problem context, we should be familiar with the
  - Domain vocabulary – terminologies meaningful to the problem
    - "Credit", "Debit", "Account" and "Statement" all have a particular meaning in the banking domain
  - Activities, processes and tasks that require software solutions
    - Credit card application
    - Home loan process
    - Enrolment process

Actors

- People/things that will interact with the software that we plan to develop?
- Actors can have an impact on how the new software should operate
- Important to analyse these impacts when designing software solutions
  - Pain points of the clients/users must be addressed
- Every actor has different pain points
  - An employee dislikes complicated and confusing systems
  - A manager needs various reports for different purposes

Actor Goals

- Every actor has a set of pain/pleasure points – we build software to address these points
- Actors also have a 'GOAL' state in mind
  - Actors use systems to reach their 'GOAL' state
- Example GOALS:
  - Track time spent working on a project
  - Get real-time access to sales information
- Software must allow the actor to achieve their 'GOALS'

Question for you

- User vs Actor vs Stakeholder

Constraints

- Constraints restrict the choices on how problems can be solved
- Example:
  - The problem must be solved in 2 months with $X
  - Improve the performance of the current system which was written in the Cobol language
  - The developed system must work in the client's operating environment, which is Linux
- Some other constraints could be
  - Laws/regulations
  - User's current training/education level

Problem Domain

- Identify actors
- Understand the need/impact of each actor
- Write down the goals of each actor
- What are the constraints that we have to work with?
- What is the domain vocabulary?

Analysing the problem to figure out a direction of solving this problem ("solution direction")

Solution Development

- Choose the best solution out of potential solutions (options/alternatives) →requires decision making
- Decision making is influenced by
  - Constraints (including some environmental factors)
  - Habits and personality
- Questions to ask
  - Are these reasonable choices?
    - Can we defend the decisions/choices that we have made?
  - What is the impact of each choice?
    - Constraints of our next steps

Assumptions

- When we develop solutions, we often make assumptions on the problem domain
- Example: The users understand English, so the instructions given to the users are written in English
- Should we verify the validity of our assumptions?
  - Better to do so, otherwise we may waste our time in developing useless solutions for the clients/users

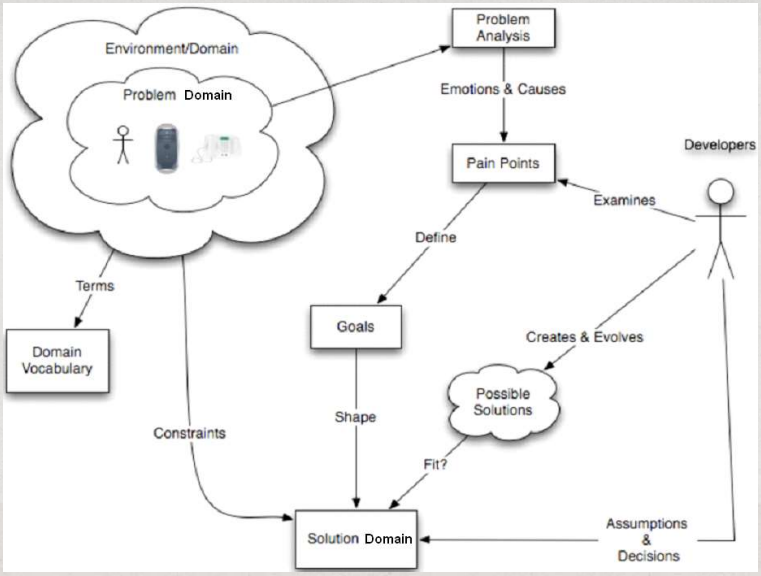
Solution Domain

- Initially, we may just have a solution direction
- Throughout the software development process, we keep refining the solutions based on
  - Our understanding of the problem domain
  - Previous decisions made for the solution
    - Note: Some of the decisions we made in the past will restrict what we do next

Problem space vs Solution space

- There is a difference between problem space and solution space
  - Software development projects start by looking at the problem space and use the information from the solution space to set boundaries
- Consider the following scenario
  - *"Mrs. Duke wants to automate all of her petrol stations with the latest gadgetry. She is rich and money is not an issue. She wants to have the sales information in real-time, esp. sales of chocolate bars".*
  - What is the pain point?
  - What attributes of the solution can we isolate?

Problem space vs Solution space

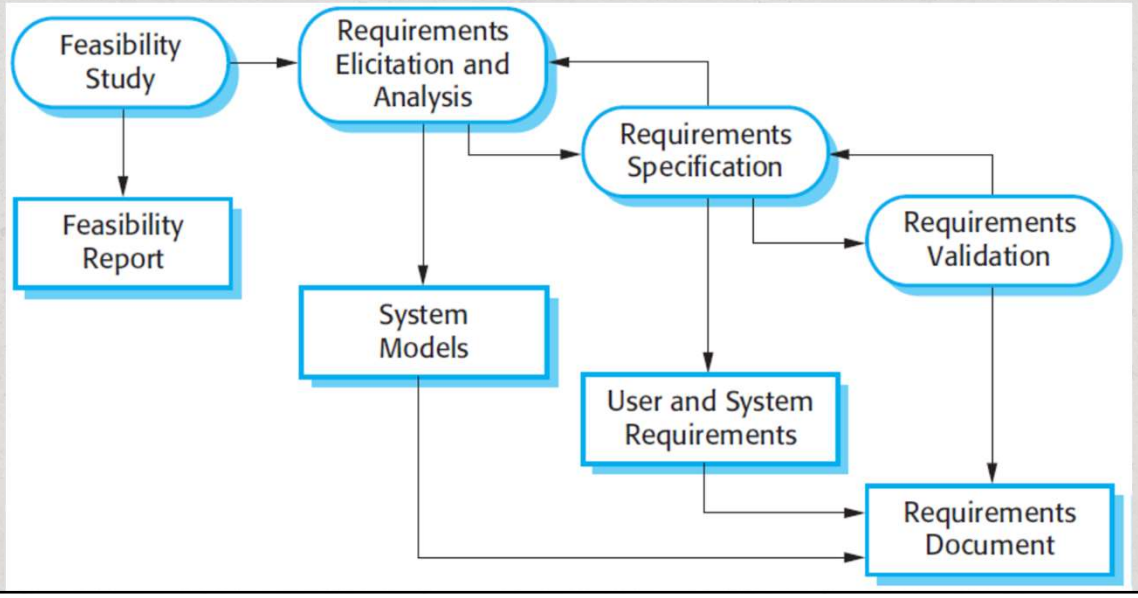


Software Requirements

Requirements Engineering

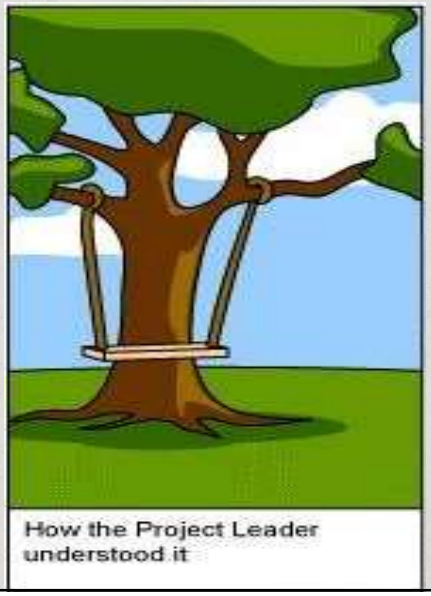
- A requirement is a feature/constraint that the client expects the system to have
- Requirements engineering aims to define the requirements of the system under construction

Requirements Engineering



Customer's Version of Requirements

Leader's Understanding of Requirements



How Analyst Designed

How Developers Implement Requirements

25

How Business Consultants Understand Requirements

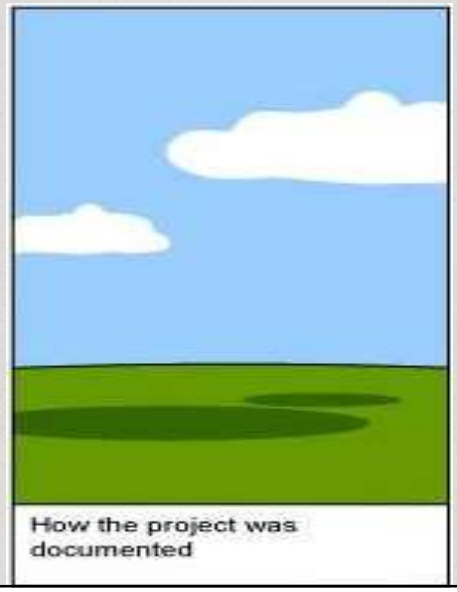
26

How Project was Documented

!!!

Here please remember how you documented your projects so far and please do not forget this figure when writing project reports!

!!!



Software on Site Installation

What Customer Pays for

29

What the Real Need was

30

Requirements engineers' feelings

What are Requirements?

- Describing what the system should do
- Wide range:
  - high-level abstract statement of a service
  - a system constraint
  - detailed mathematical functional specification

What are Requirements?

- 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.

What are Requirements?

"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."

What are Requirements?

- Something required, something wanted or needed
- There is a huge difference between wanted and needed and it should be kept in mind all the time
  - Need – something you have to have
  - Want – something you would like to have

Levels and Types of Requirements

- Levels
  - User requirements: statements
  - System requirements: detailed description
- Types
  - Functional: services
  - Non-functional: constraints

Functional vs. Non-Functional Requirements

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

Functional vs. Non-Functional Requirements

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

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.

Functional Requirements

SatWatch is a wrist watch that displays the time based on its current location. SatWatch uses GPS satellites (Global Positioning System) to determine its location and internal data structures to convert this location into a time zone.

The information stored in SatWatch and its accuracy measuring time is such that the watch owner never needs to reset the time. SatWatch adjusts the time and date displayed as the watch owner crosses time zones and political boundaries. For this reason, SatWatch has no buttons or controls available to the user.

SatWatch determines its location using GPS satellites and, as such, suffers from the same limitations as all other GPS devices (e.g., inability to determine location at certain times of the day in mountainous regions). During blackout periods, SatWatch assumes that it does not cross a time zone or a political boundary. SatWatch corrects its time zone as soon as a blackout period ends.

SatWatch has a two-line display showing, on the top line, the time (hour, minute, second, time zone) and on the bottom line, the date (day, date, month, year). The display technology used is such that the watch owner can see the time and date even under poor light conditions.

When political boundaries change, the watch owner may upgrade the software of the watch using the WebifyWatch device (provided with the watch) and a personal computer connected to the Internet.

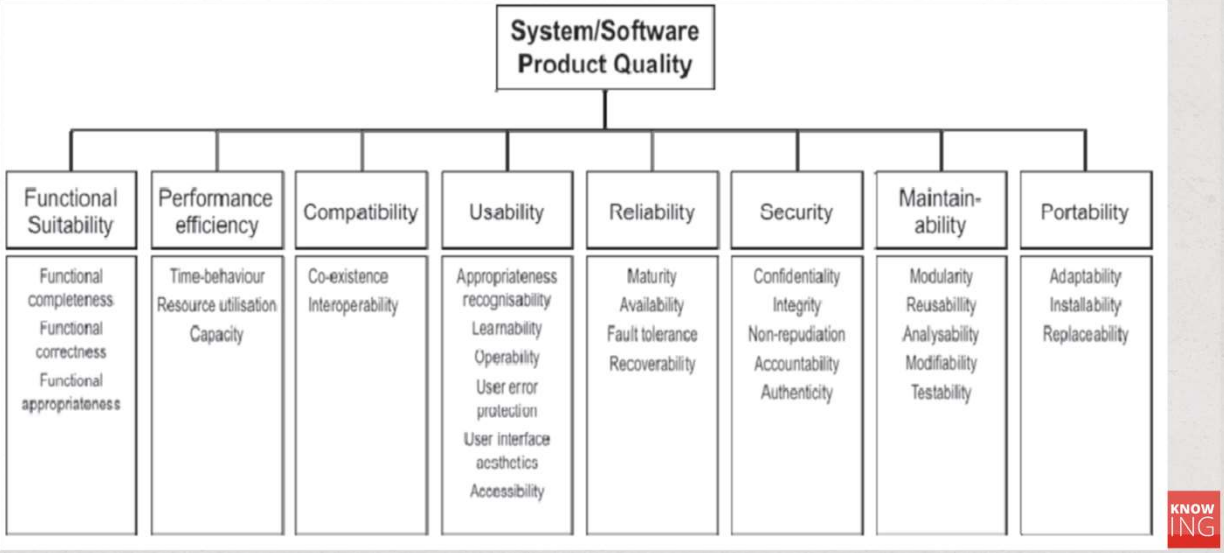
Non-Functional Requirements

- 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 Integrated Development Environment (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.

Non-Functional Requirements

- Non-functional requirements may affect the overall architecture of a system rather than the individual 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.

Quality characteristics



Functional Suitability

- Degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions
  - Functional completeness: degree to which the set of functions covers all the specified tasks and user objectives
  - Functional correctness: degree to which a product or system provides the correct results with the needed degree of precision
  - Functional appropriateness: degree to which the functions facilitate the accomplishment of specified tasks and objectives

Functional Suitability – Discussion

- Different from functional requirement!!!
  - Functional completeness: all requirements documented & other objectives met
  - Functional correctness: any part of system with exact and precise output
  - Functional appropriateness: overall aim – meet the business need

Performance Efficiency

- Performance relative to the amount of resources used under stated conditions
  - Time behaviour: degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements
  - Resource utilization: degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements
  - Capacity: degree to which the maximum limits of a product or system parameter meet requirements

Compatibility

- Degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment
  - Co-existence: degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product
  - Interoperability: degree to which two or more systems, products or components can exchange information and use the information that has been exchanged

Usability

- Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use
  - Appropriateness recognisability: degree to which users can recognise whether a product or system is appropriate for their needs
  - Learnability: degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use

Usability (cont'd)

- Operability: degree to which a product or system has attributes that make it easy to operate and control
- User error protection: degree to which a system protects users against making errors
- User interface aesthetics: degree to which a user interface enables pleasing and satisfying interaction for the user
- Accessibility: degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use

Reliability

- Degree to which a system, product or component performs specified functions under specified conditions for a specified period of time
  - Maturity: degree to which a system, product or component meets needs for reliability under normal operation
  - Availability: degree to which a system, product or component is operational and accessible when required for use

Reliability (cont'd)

- Fault tolerance: degree to which a system, product or component operates as intended despite the presence of hardware or software faults
- Recoverability: degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system

Security

- Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization
  - Confidentiality: degree to which a product or system ensures that data are accessible only to those authorized to have access
  - Integrity: degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data

Security (cont'd)

- Non-repudiation: degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later
- Accountability: degree to which the actions of an entity can be traced uniquely to the entity
- Authenticity: degree to which the identity of a subject or resource can be proved to be the one claimed

Maintainability

- Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers
  - Modularity: degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components
  - Reusability: degree to which an asset can be used in more than one system, or in building other assets

Maintainability (cont'd)

- Analysability: degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified
- Modifiability: degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality
- Testability: degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met

Portability

- Degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another
  - Adaptability: degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments

Portability (cont'd)

- Installability: degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment
- Replaceability: degree to which a product can replace another specified software product for the same purpose in the same environment

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 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

Identifying Non-Functional Requirements

Quality requirements for SatWatch

- Any user who knows how to read a digital watch and understands international time zone abbreviations should be able to use SatWatch without the user manual. [Usability requirement]
- As the SatWatch has no buttons, no software faults requiring the resetting of the watch should occur. [Reliability requirement]
- SatWatch should display the correct time zone within 5 minutes of the end of a GPS blackout period. [Performance requirement]
- SatWatch should measure time within 1/100th second over 5 years. [Performance requirement]
- SatWatch should display time correctly in all 24 time zones. [Performance requirement]
- SatWatch should accept upgrades to its onboard via the Webify Watch serial interface. [Supportability requirement]

Constraints for SatWatch

- All related software associated with SatWatch, including the onboard software, will be written using Java, to comply with current company policy. [Implementation requirement]
- SatWatch complies with the physical, electrical, and software interfaces defined by WebifyWatch API 2.0. [Interface requirement]

Identifying Non-Functional Requirements

- How reliable, available, and robust should the system be?
- Is restarting the system acceptable in the event of a failure?
- How much data can the system loose?
- How should the system handle exceptions?
- Are there safety requirements of the system?
- Are there security requirements of the system?

Identifying Non-Functional Requirements

- How responsive should the system be?
- Are any user tasks time critical?
- How many concurrent users should it support?
- How large is a typical data store for comparable systems?
- What is the worst latency that is acceptable to users?
- What is the level of expertise of the user?
- What user interface standards are familiar to the user?
- What documentation should be provided to the user?

Good/Bad Non-Functional Requirements

- "The system must be usable."
- "The system must provide visual feedback to the user within one second of issuing a command."
- "The availability of the system must be above 95 percent."
- "The user interface of the new system should be similar enough to the old system that users familiar with the old system can be easily trained to use the new system."

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, it is impossible to produce a complete and consistent requirements document.

Unambiguity and Correctness

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Requirements should be correct in terms of representing accurately the system that clients need.

Other Desired Properties of Requirements

- Realism
  - Can be implemented within constraints
- Verifiability
  - Repeatable test cases
- Traceability
  - Each requirement can be traced throughout the development

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.

Is It Necessary?

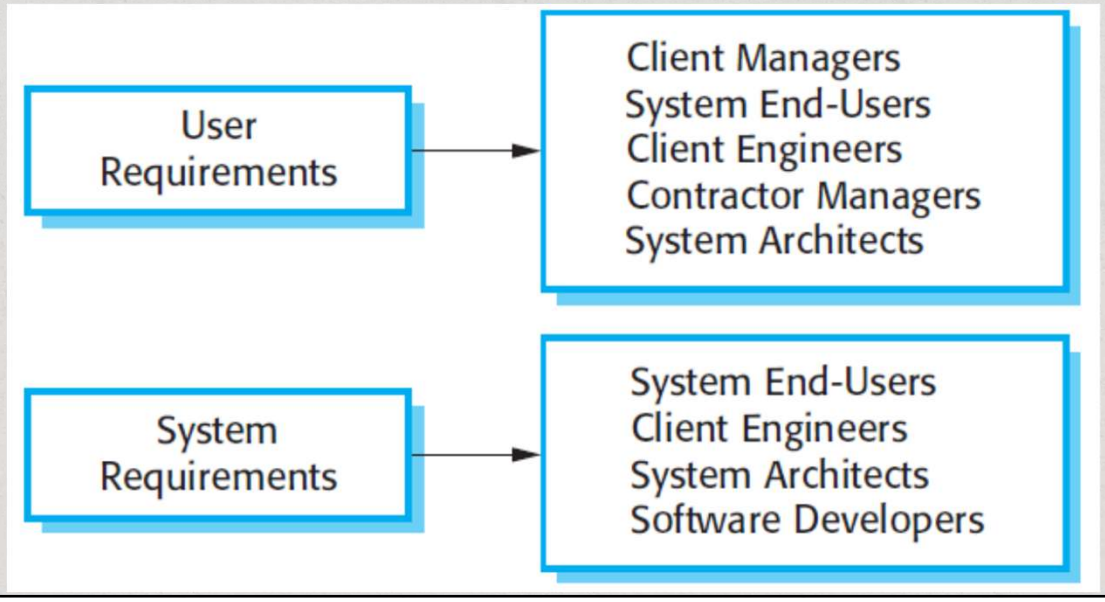
Yes!

- As a reference point regarding the project
- Provides information about the project
- Serves as reference for preparing other project documents

Levels of Requirements

- 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.

Readers



69

Usage

System Customers	Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers	Use the requirements document to plan a bid for the system and to plan the system development process.
System Engineers	Use the requirements to understand what system is to be developed.
System Test Engineers	Use the requirements to develop validation tests for the system.
System Maintenance Engineers	Use the requirements to understand the system and the relationships between its parts.

70

Structure of 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.

Structure of 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 Specification

- Process of writing down the user and system requirements
- User requirements:
  - Functional and non-functional requirements
- System requirements:
  - Expanded version of user requirements

Requirements vs. 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.

Requirements Specification

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 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.

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

Structured Specification

- 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.

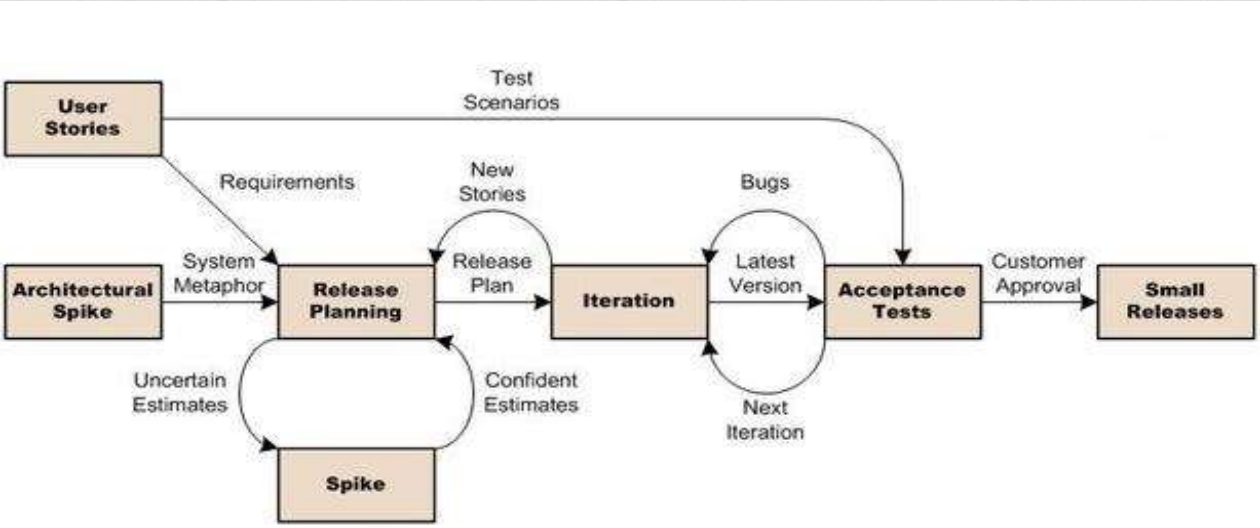
Standard Form

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

User Story

- User stories are more than software system requirements
  - Putting actual users at the centre of conversation
- Users tell the stories; developers listen, ask questions to understand context
  - A short description of the *behaviour* of the system – generally implementation issues are not discussed
  - Smallest unit of work that delivers value to the customer
- The entire system is specified through stories
- Each story is short, goal-oriented – *testable*
  - Can be used for usability testing

Agile Development – eXtreme Programming



81

What is a User Story?

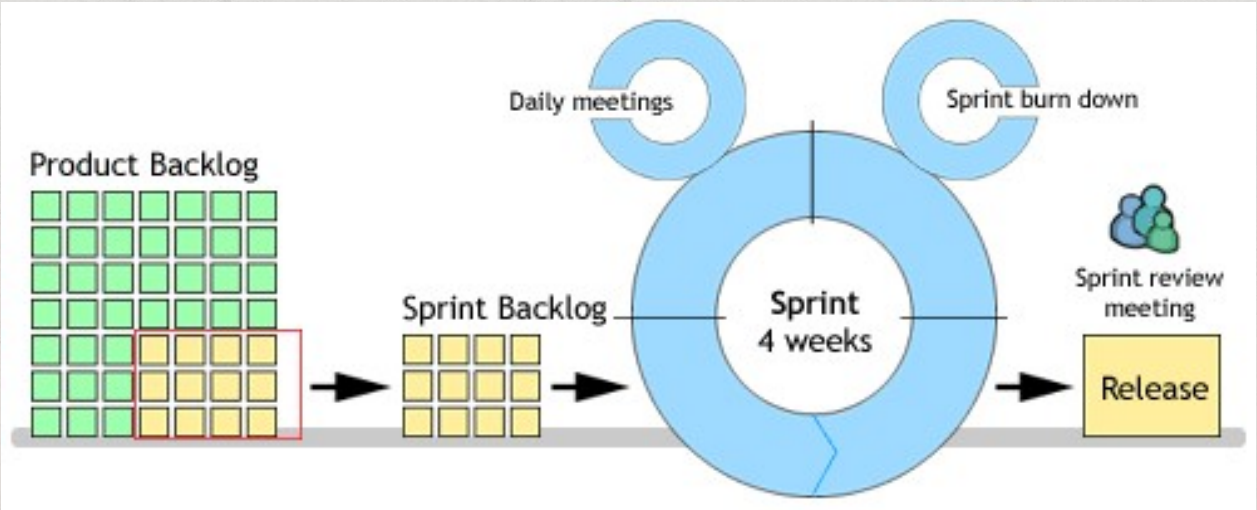
82

User Story – An Example

- Example: the parking fee system
- User story:

  As a car park manager, I want to develop a system so that the parking fee can be calculated according to the information including type of vehicle, day of week, and hours of parking.

Agile Development – Scrum



Creating a sprint backlog

Example Product Backlog

ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		**30**	

Source: https://www.scrum-institute.org/The_Scrum_Product_Backlog.php

Summary

- Importance of requirements
- Requirements document
- Requirements specification
- Desired properties of requirements

References

- Sommerville, I. *Software Engineering* (Chapter 4)
- ISO/IEC/IEEE 29148. *Systems and software engineering — Life cycle processes — Requirements engineering*
- ISO/IEC 25010:2011. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*
- Bruegge, B. & Dutoit A.H. *Object-Oriented Software Engineering Using UML, Patterns, and Java* (Chapter 4)

Next

- Tutorial
  - Writing requirements
- Next week
  - Modelling and Design