ICT 4203
Software Engineering

Software Engineering and Crisis

Prasadini Padmasiri

What is a Software?

- **Software** is a program or set of programs containing instructions that provide desired functionality.

- **Engineering** is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

What is Software Engineering?

- **Software Engineering** is the process of designing, developing, testing, and maintaining software.

- It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.

What is Software Engineering?

1. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.
2. It is a rapidly evolving field, and new tools and technologies are constantly being developed to improve the software development process.
3. By following the principles of software engineering and using the appropriate tools and methodologies, software developers can create high-quality, reliable, and maintainable software that meets the needs of its users.
4. Software Engineering is mainly used for large projects based on software systems rather than single programs or applications.
5. The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency.
6. Software Engineering ensures that the software that has to be built should be consistent, correct, also on budget, on time, and within the required requirements.

What is Software Engineering?

1. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.
2. It is a rapidly evolving field, and new tools and technologies are constantly being developed to improve the software development process.
3. By following the principles of software engineering and using the appropriate tools and methodologies, software developers can create high-quality, reliable, and maintainable software that meets the needs of its users.
4. Software Engineering is mainly used for large projects based on software systems rather than single programs or applications.
5. The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency.
6. Software Engineering ensures that the software that has to be built should be consistent, correct, also on budget, on time, and within the required requirements.

Key Principles of Software Engineering

- **Modularity**: Breaking the software into smaller, reusable components that can be developed and tested independently.
- **Abstraction**: Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.
- **Encapsulation**: Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
- **Reusability**: Creating components that can be used in multiple projects, which can save time and resources.

Key Principles of Software Engineering

- **Maintenance**: Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
- **Testing**: Verifying that the software meets its requirements and is free of bugs.
- **Design Patterns**: Solving recurring problems in software design by providing templates for solving them.
- **Agile methodologies:** Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery, and flexibility.
- **Continuous Integration & Deployment:** Continuously integrating the code changes and deploying them into the production environment.

Main Attributes of Software Engineering

- **Efficiency:** It provides a measure of the resource requirement of a software product in an efficient way.
- **Reliability:** It provides the assurance that the product will deliver the same results when used in similar working environment.
- **Reusability:** This attribute makes sure that the module can be used in multiple applications.
- **Maintainability:** It is the ability of the software to be modified, repaired, or enhanced easily with changing requirements.

Dual Role of Software

- There is a dual role of software in the industry. The first one is as **a product** and the other one is as **a vehicle for delivering the product**.

Dual Role of Software

1. As a Product

- It delivers computing potential across networks of Hardware.
- It enables the Hardware to deliver the expected functionality.
- It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

Dual Role of Software

2. As a Vehicle for Delivering a Product

- It provides system functionality (e.g., payroll system).
- It controls other software (e.g., an operating system).
- It helps build other software (e.g., software tools).

Objectives of Software Engineering

- Maintainability
- Efficiency
- Correctness
- Reusability
- Testability
- Reliability
- Portability
- Adaptability
- Interoperability

Program vs Software Product

Parameters	Program	Software Product
Definition	A program is a set of instructions that are given to a computer in order to achieve a specific task.	Software is when a program is made available for commercial business and is properly documented along with its licensing. **Software Product = Program + Documentation + Licensing.**
Stages Involved	Program is one of the stages involved in the development of the software.	Software Development usually follows a life cycle, which involves the feasibility study of the project, requirement gathering, development of a prototype, system design, coding, and testing.

Advantages of Software Engineering

- Improved Quality
- Increased Productivity
- Better Maintainability
- Reduced Costs
- Increased Customer Satisfaction
- Better Team Collaboration
- Better Scalability
- Better Security

Disadvantages of Software Engineering

- High upfront costs
- Limited flexibility
- Bureaucratic
- Complexity
- Limited creativity
- High learning curve
- High dependence on tools
- High maintenance

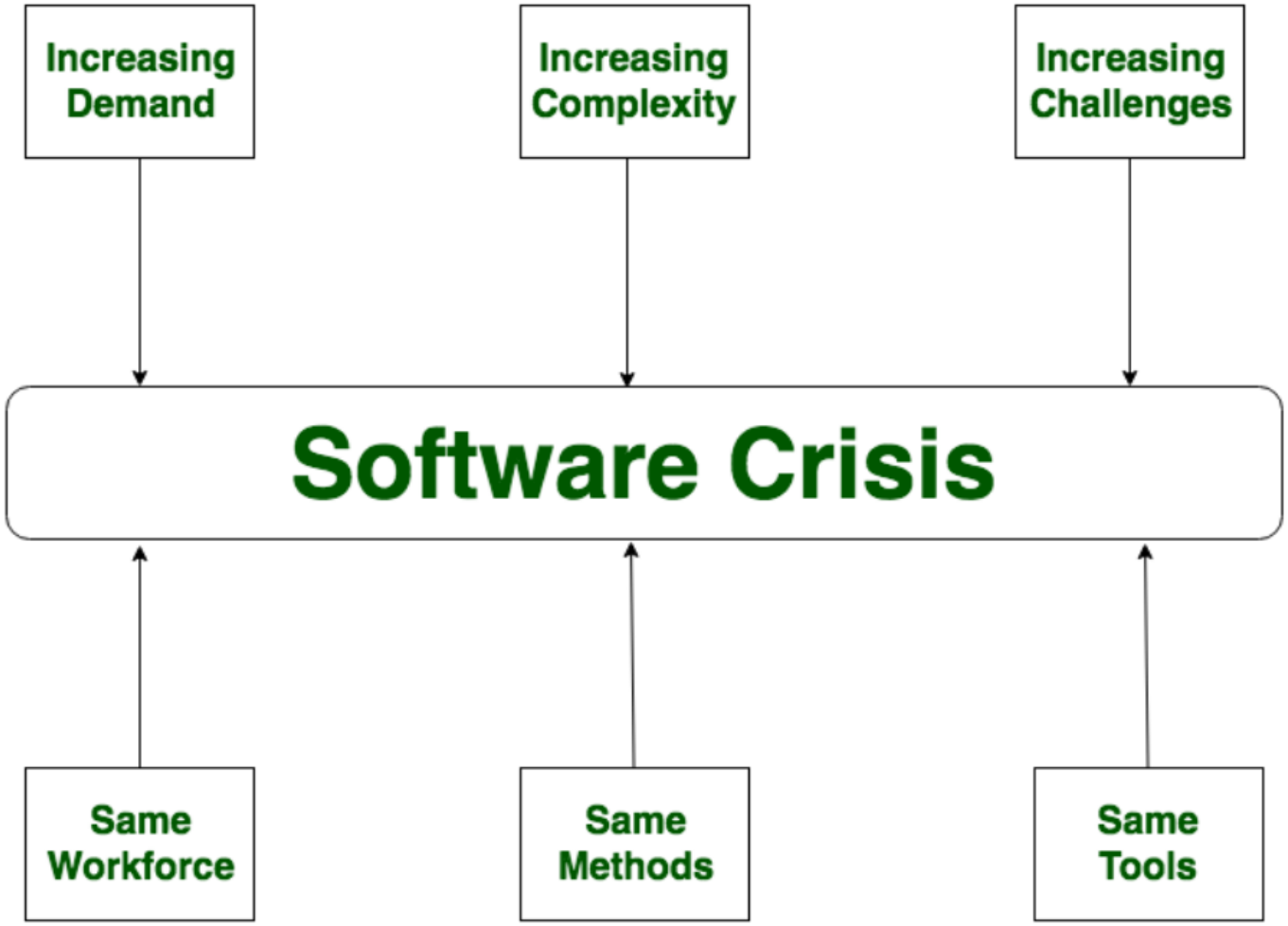
Software Crisis

Software Crisis

- Software Crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time.

- The software crisis was due to using **the same workforce, same methods, and same tools even though rapidly increasing software demand, the complexity of software, and software challenges**.

- With the increase in software complexity, many software problems arise because existing methods were insufficient.


Increasing Demand
Increasing Complexity
Increasing Challenges
Software Crisis
Same Workforce
Same Methods
Same Tools

Causes of Software Crisis

- The cost of owning and maintaining software was as expensive as developing the software.
- At that time Projects were running overtime.
- At that time Software was very inefficient.
- The quality of the software was low quality.
- Software often did not meet user requirements.
- The average software project overshoots its schedule by half.
- At that time Software was never delivered.
- Non-optimal resource utilization.
- Challenging to alter, debug, and enhance.
- The software complexity is harder to change.

Factors Contributing to Software Crisis

- Poor project management.
- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

Solution of Software Crisis

- Reduction in software over budget.
- The quality of the software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working on the software project.
- Software must be delivered.
- Software must meet user requirements.

Thank you

ICT 4203
Software Engineering

Prasadini Padmasiri

1. What is a software?

Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.

1. Generic products

- These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them.
- Examples of this type of product include software for PCs such as databases, word processors, drawing packages, and project-management tools.

2. Customized (or bespoke) products

- These are systems that are commissioned by a particular customer.
- A software contractor develops the software especially for that customer.
- Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

2. What are the attributes of good software?

Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.

3. What is software engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production.

4. What are the fundamental software engineering activities?

Software specification, software development, software validation, and software evolution.

Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.

Software development, where the software is designed and programmed.

Software validation, where the software is checked to ensure that it is what the customer requires.

Software evolution, where the software is modified to reflect changing customer and market requirements.

5. What is the difference between software engineering and computer science?

Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

6. What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.

7. What are the key challenges facing software engineering?

Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.

8. What are the costs of software engineering?

Roughly 60% of software costs are development costs; 40% are testing costs. For custom software, evolution costs often exceed development costs.

9. What are the best software engineering techniques and methods?

While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.

10. What differences has the Web made to software engineering?

The Web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse

Thank you

ICT 4203
Software Engineering

Software Specification – Part 01

Requirements

Prasadini Padmasiri

Software Process Activities

- Software specification
- Software development
- Software validation
- Software evolution

Software specification - The functionality of the software and constraints on its operation must be defined.

Software development - The software to meet the specification must be produced.

Software validation - The software must be validated to ensure that it does what the customer wants.

Software evolution - The software must evolve to meet changing customer needs.

Software specification - The functionality of the software and constraints on its operation must be defined.

Software development - The software to meet the specification must be produced.

Software validation - The software must be validated to ensure that it does what the customer wants.

Software evolution - The software must evolve to meet changing customer needs.

Software Specification

- **Software Specification** is the process of defining the objectives, functionalities, and constraints of a software system.
- It involves understanding and documenting what the software should do, identifying the needs of stakeholders, and creating detailed requirements.
- This activity includes
    - **Requirements elicitation**, where information is gathered from stakeholders;
    - **Requirements analysis**, where conflicts are resolved, and priorities are set; and
    - **Requirements documentation**, which formalizes these into a software requirements specification (SRS). Finally, it includes
    - **Requirements validation** to ensure they accurately reflect stakeholder expectations and are feasible for implementation.

What is a Requirement?

- The requirements for a system are the descriptions of what the system should do—the services that it provides and the constraints on its operation.

- These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.

- The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering** (RE).

User Requirements

- User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

System Requirements

- System requirements are more detailed descriptions of the software system's functions, services, and operational constraints.

- The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented.

- It may be part of the contract between the system buyer and the software developers.

User requirements definition

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

System requirements specification

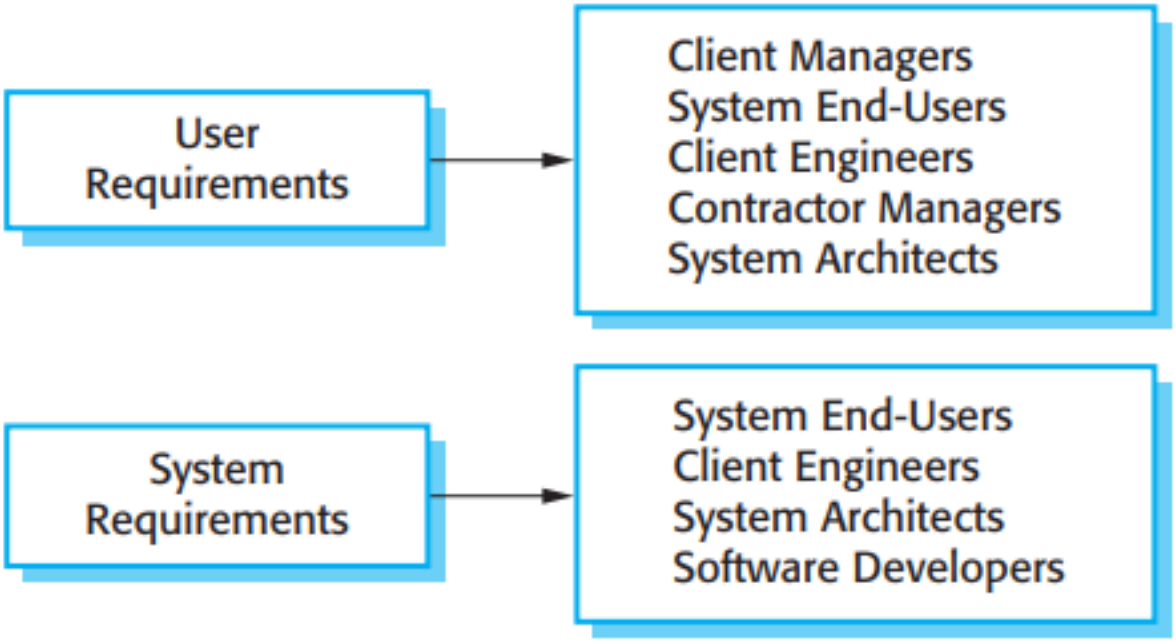
1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list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

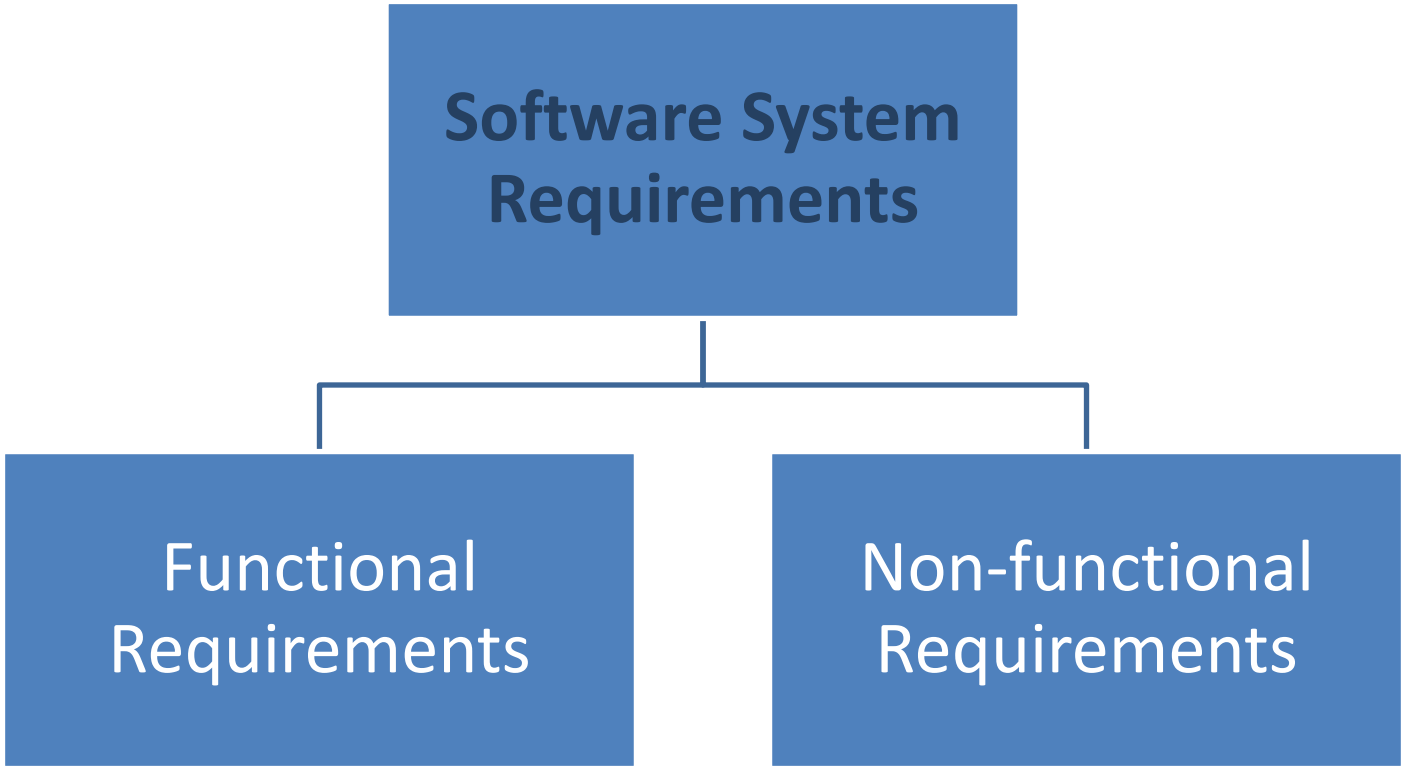
Readers of the Requirements

- The readers of the user requirements are not usually concerned with how the system will be implemented and may be managers who are not interested in the detailed facilities of the system.

- The readers of the system requirements need to know more precisely what the system will do because they are concerned with how it will support the business processes or because they are involved in the system implementation.

Readers of the Requirements




Software System Requirements
Functional Requirements
Non-functional Requirements

Functional Requirements

- The functional requirements for a system describe what the system should do.
- These requirements depend on the type of software being developed, the expected users of the software, and the general approach taken by the organization when writing requirements.
- When expressed as user requirements, functional requirements are usually described in an abstract way that can be understood by system users.
- However, more specific functional system requirements describe the system functions, its inputs and outputs, exceptions, etc., in detail.

- Examples for functional requirements for the Mentcare system, used to maintain information about patients receiving treatment for mental health problems:

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

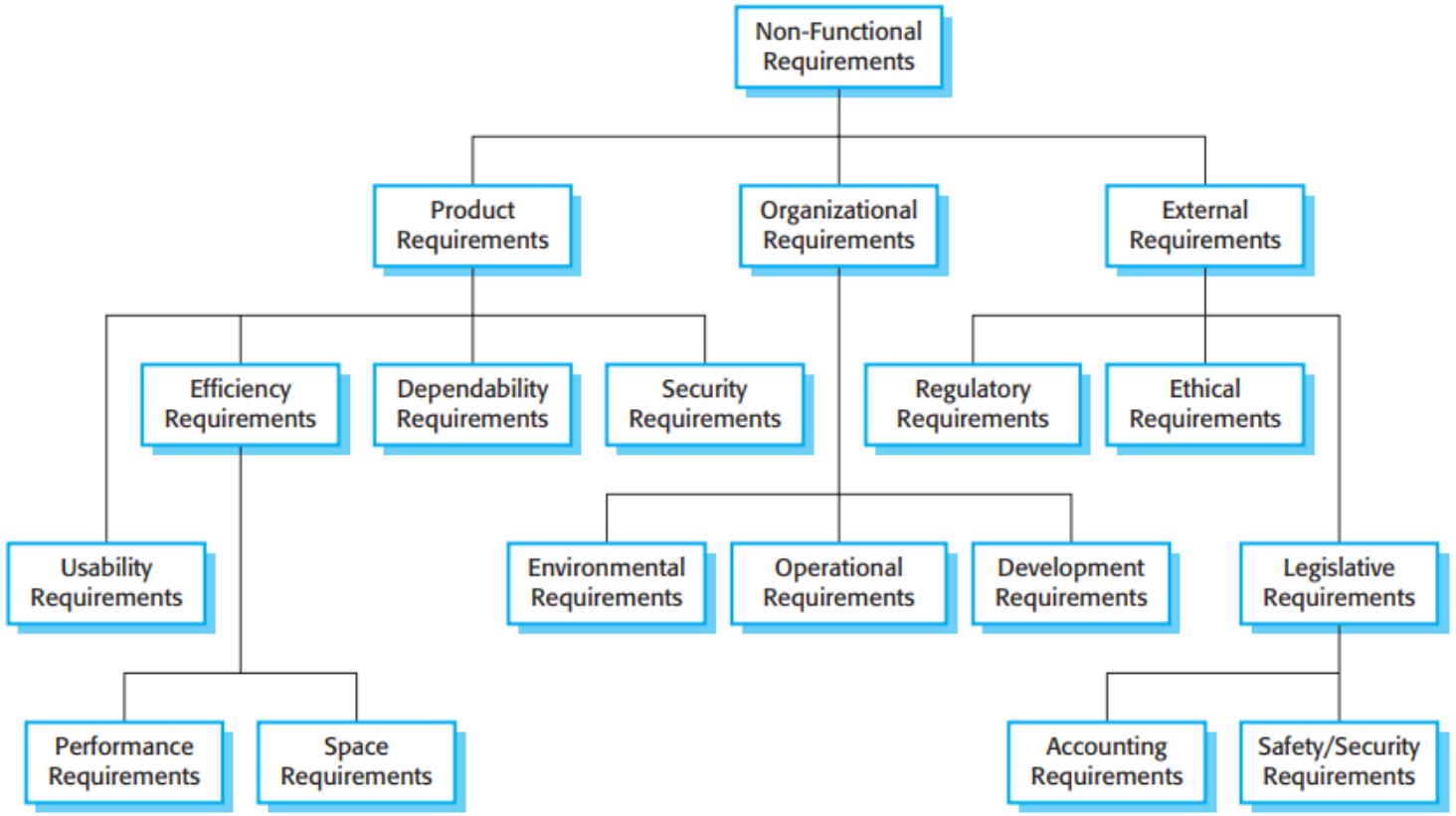
Non-functional Requirements

- Non-functional requirements are requirements that are not directly concerned with the specific services delivered by the system to its users.
- They may relate to emergent system properties such as reliability, response time, and store occupancy.
- Alternatively, they may define constraints on the system implementation such as the capabilities of I/O devices or the data representations used in interfaces with other systems.
- Non-functional requirements, such as performance, security, or availability, usually specify or constrain characteristics of the system as a whole.

Non-functional Requirements

- Non-functional requirements are often more critical than individual functional requirements.
- System users can usually find ways to work around a system function that doesn't really meet their needs.
- However, failing to meet a non-functional requirement can mean that the 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; if an embedded control system fails to meet its performance requirements, the control functions will not operate correctly

Types of Non-functional Requirements



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

Example: **Medical staff shall be able to use all the system functions after two hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use**.

Differences between Functional and Non - functional Requirements

Functional Requirement	Non-functional Requirement
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.

Examples

Functional Requirement	Non-functional Requirement
1. The system should authenticate the user whenever he/she logs into the system.	1. Emails should be sent with a latency of no greater than 12 hours from such an activity.
2. The system should shutdown in case of a cyber attack.	2. The processing of each request should be done within 10 seconds
3. The user should receive a Verification email whenever he/she registers for the first time on some software system.	3. The site should load in 3 seconds when the number of simultaneous users are > 10000

In Class Activity

Consider the following scenario and answer the questions below.

‘Perfect Gift’ is an online gift-ordering web application for customers. the registered customers are allowed to order gifts through the application. Customers can order gifts by clicking on any of the items shown on the web page.

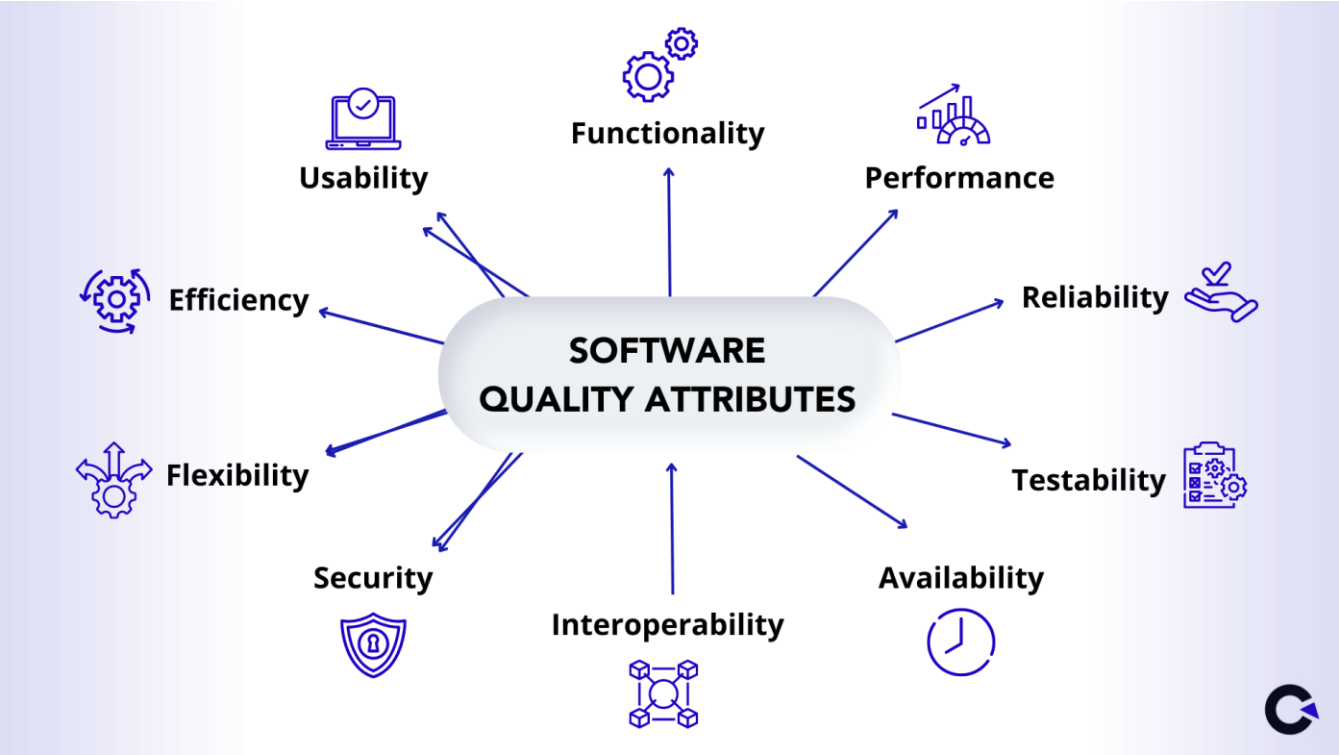
To order gifts, first customers have to log in to the application. Once the customer enters order details such as the number of gifts and the delivery address, the system asks the customer to enter the credit card details. The credit card details together with the transaction details are sent to the payment gateway (PG). It validates the received details and sends a confirmation back to the web application.

To maintain a smooth delivery, gift suppliers are also given accounts in the web application. When the customer purchases gifts, the suppliers are notified and suppliers in return confirm the order. For example, if a customer buys a iPhone, the supplier of the phone gets notified and he will confirm the order to the web application.

In addition to the management of the gift, another main source of income for web application is managing advertisements. The administrator updates the web application with the advertisements he receives. The manager goes through the advertisements and approves them for publishing. The application will internally count the number of hits (number of users who have viewed/clicked on the advertisement). The administrator gels a weekly statistics report on hit count of the advertisements.

1. Write down four functional requirements of the system.
2. Write down four non-functional requirements of the system.

Software Quality Attributes



Quality Attributes

- **Maintainability** - the degree of efficiency to which a software solution can be modified for its improvement or adaptation to the evolving requirements or changes in the environment.

- **Reliability** – the degree to which a software system or its components performs specific functions under predefined conditions for a certain period of time

- **Usability** - the ease with which users can perform a specific task on the system

- **Interoperability** – ability to communicate or exchange data seamlessly between different operating systems, databases, and protocol conditions

- **Security** - the degree to which a software system safeguards the information or data

Quality Attributes

- **Testability** - how easy a software solution is to test to find bugs or ensure that it meets all predefined criteria.
- **Reusability** - the degree to which software components can be reused in another application or the same application.
- **Supportability** - the degree to which a software system can provide useful information for identifying and resolving the issues when application/functionality stops working.
- **Scalability** - the ability of a software system to handle the increased load without decreasing its performance.

https://testsigma.com/blog/software-quality-attributes/
https://syndicode.com/blog/12-software-architecture-quality-attributes/

Thank you

ICT 4203
Software Engineering

Software Specification – Part 02

Requirement Elicitation Techniques

Prasadini Padmasiri

Software Process Activities

- Software specification
- Software development
- Software validation
- Software evolution

Software specification - The functionality of the software and constraints on its operation must be defined.

Software development - The software to meet the specification must be produced.

Software validation - The software must be validated to ensure that it does what the customer wants.

Software evolution - The software must evolve to meet changing customer needs.

Software specification - The functionality of the software and constraints on its operation must be defined.

Software development - The software to meet the specification must be produced.

Software validation - The software must be validated to ensure that it does what the customer wants.

Software evolution - The software must evolve to meet changing customer needs.

Software Specification

- **Software Specification** is the process of defining the objectives, functionalities, and constraints of a software system.
- It involves understanding and documenting what the software should do, identifying the needs of stakeholders, and creating detailed requirements.
- This activity includes
    - **Requirements elicitation**, where information is gathered from stakeholders;
    - **Requirements analysis**, where conflicts are resolved, and priorities are set; and
    - **Requirements documentation**, which formalizes these into a software requirements specification (SRS). Finally, it includes
    - **Requirements validation** to ensure they accurately reflect stakeholder expectations and are feasible for implementation.

Requirements Elicitation

- The aims of the requirements elicitation process are to understand the work that stakeholders do and how they might use a new system to help support that work.

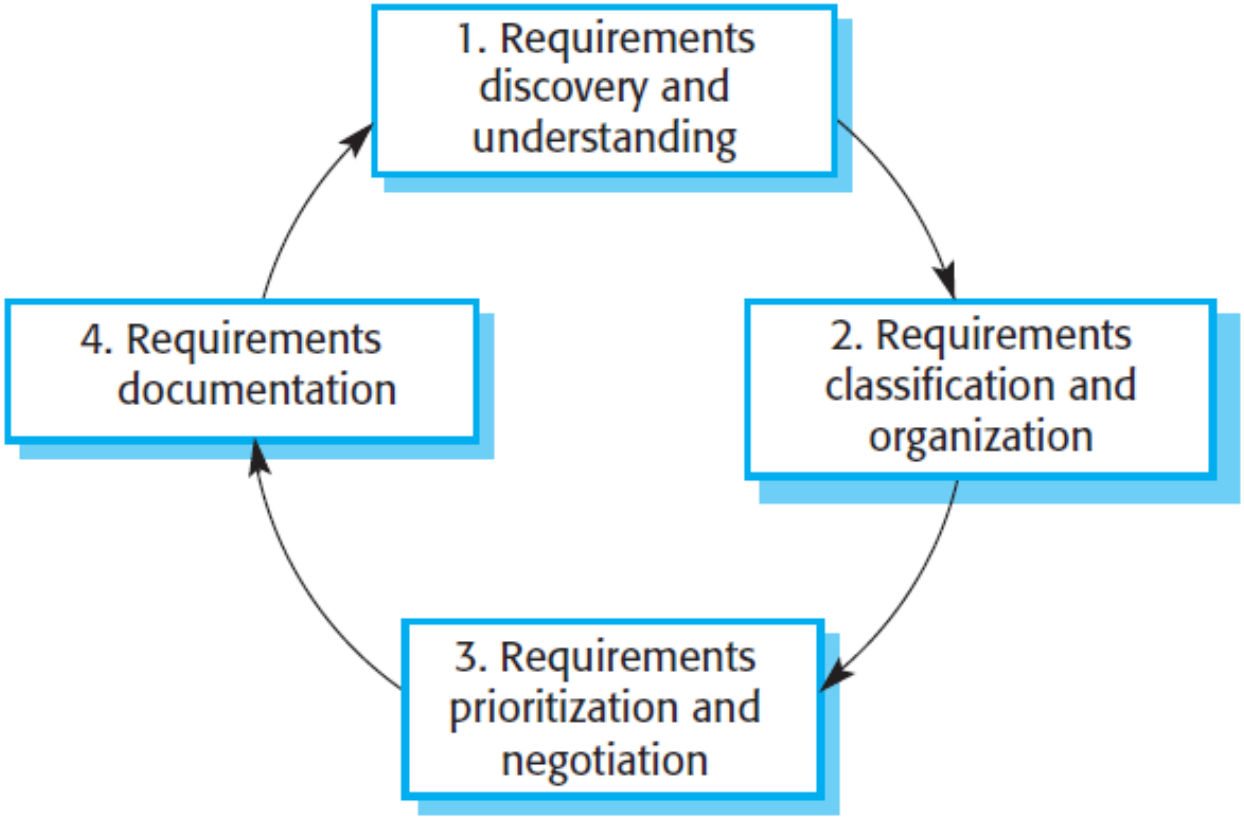
- During requirements elicitation, software engineers work with stakeholders to find out about the application domain, work activities, the services and system features that stakeholders want, the required performance of the system, hardware constraints, and so on.

Eliciting and understanding requirements from system stakeholders is a difficult process for **several reasons**:

- Stakeholders often don't know what they want from a computer system except in the most general terms;

- Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, may not understand these requirements.

- Different stakeholders, with diverse requirements, may express their requirements in different ways. Requirements engineers have to discover all potential sources of requirements and discover commonalities and conflict.


1. Requirements discovery and understanding
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements documentation

Requirements discovery and understanding

- This is the process of interacting with stakeholders of the system to discover their requirements.
- Domain requirements from stakeholders and documentation are also discovered during this activity.

Requirements classification and organization

- This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.

Requirements prioritization and negotiation

- Inevitably, when multiple stakeholders are involved, requirements will conflict.
- This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts.

Requirements documentation

- The requirements are documented and input into the next round of the spiral.
- An early draft of the software requirements documents may be produced at this stage, or the requirements may simply be maintained informally on whiteboards, wikis, or other shared spaces.

Requirement Elicitation Techniques

- Requirements elicitation involves meeting with stakeholders of different kinds to discover information about the proposed system.

- There are some fundamental approaches to requirements elicitation:
  - Interviews
  - Questionnaire/Survey
  - Observation or ethnography
  - Prototyping
  - Brainstorming Sessions
  - Joint Application Development

Interviews

- The objective of conducting an interview is to understand the customer's expectations of the software.
- It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
- Formal or informal interviews with system stakeholders are part of most requirements engineering processes.
- In these interviews, the requirements engineering team puts questions to stakeholders about the system that they currently use and the system to be developed.

Interviews

Interviews may be of two types:

- Closed interviews, where the stakeholder answers a predefined set of questions.
- Open interviews, in which there is no predefined agenda. The requirements engineering team explores a range of issues with system stakeholders and hence develops a better understanding of their needs

Interviews

- Eliciting domain knowledge through interviews can be difficult, for two reasons:

1. All application specialists use jargon specific to their area of work. It is impossible for them to discuss domain requirements without using this terminology. They normally use words in a precise and subtle way that requirements engineers may misunderstand.

2. Some domain knowledge is so familiar to stakeholders that they either find it difficult to explain or they think it is so fundamental that it isn’t worth mentioning.

Interviews

- To be an effective interviewer, you should bear two things in mind:

1. You should be open-minded, avoid preconceived ideas about the requirements, and willing to listen to stakeholders. If the stakeholder comes up with surprising requirements, then you should be willing to change your mind about the system.

2. You should prompt the interviewee to get discussions going by using a springboard question or a requirements proposal, or by working together on a prototype system. Saying to people “tell me what you want” is unlikely to result in useful information. They find it much easier to talk in a defined context rather than in general terms.

Ethnography

- Ethnography is an observational technique that can be used to understand operational processes and help derive requirements for software to support these processes.
- An analyst immerses himself or herself in the working environment where the system will be used.
- The day-to-day work is observed, and notes are made of the actual tasks in which participants are involved.
- The value of ethnography is that it helps discover implicit system requirements that reflect the actual ways that people work, rather than the formal processes defined by the organization.

Ethnography

- Ethnography is particularly effective for discovering two types of requirements:

1. Requirements derived from the way in which people actually work, rather than the way in which business process definitions say they ought to work. In practice, people never follow formal processes.

2. Requirements derived from cooperation and awareness of other people's activities.

Brainstorming Session

- Brainstorming Sessions is a group technique.
- It is intended to generate lots of new ideas hence providing a platform to share views.
- A highly trained facilitator is required to handle group bias and conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

Prototyping

- Prototyping is a relatively modern technique for gathering requirements. In this approach, you gather preliminary requirements that you use to build an initial version of the solution - a prototype.

- You show this to the client, who then gives you additional requirements. You change the application and cycle around with the client again.

- This repetitive process continues until the product meets the critical mass of business needs or for an agreed number of iterations.

Prototyping

- Prototyping involves construction of models of the system in order to discover new features.
- Prototypes can involve **working models and nonworking models**.
  - *Working models* can include executable code in the case of software systems and simulations, or temporary or to-scale prototypes for non-software systems.
  - *Nonworking models* can include storyboards and mock-ups of user interfaces.

Questionnaire/Survey

- When collecting information from many people – too many to interview with budget and time constraints – a survey or questionnaire can be used.

- Survey questions of any type can be used. For example, questions can be **closed** (ex: multiple choice, true-false) or **open ended**, involving free-form responses.

  - *Closed questions* have the advantage of easier coding for analysis, and they help to bind the scope of the system.

  - *Open questions* allow for more freedom and innovation, but can be harder to analyze and can encourage scope creep.

Document Analysis

- Reviewing the documentation of an existing system can help when creating AS–IS process document, as well as driving gap analysis for scoping of migration projects.

- In an ideal world, we would even be reviewing the requirements that drove creation of the existing system – a starting point for documenting current requirements.

- Nuggets of information are often buried in existing documents that help us ask questions as part of validating requirement completeness.

Scenarios

- Scenarios are informal descriptions of the system in use that provide a high-level description of system operation, classes of users, and exceptional situations.

- Here is a sample scenario for the pet store POS system.

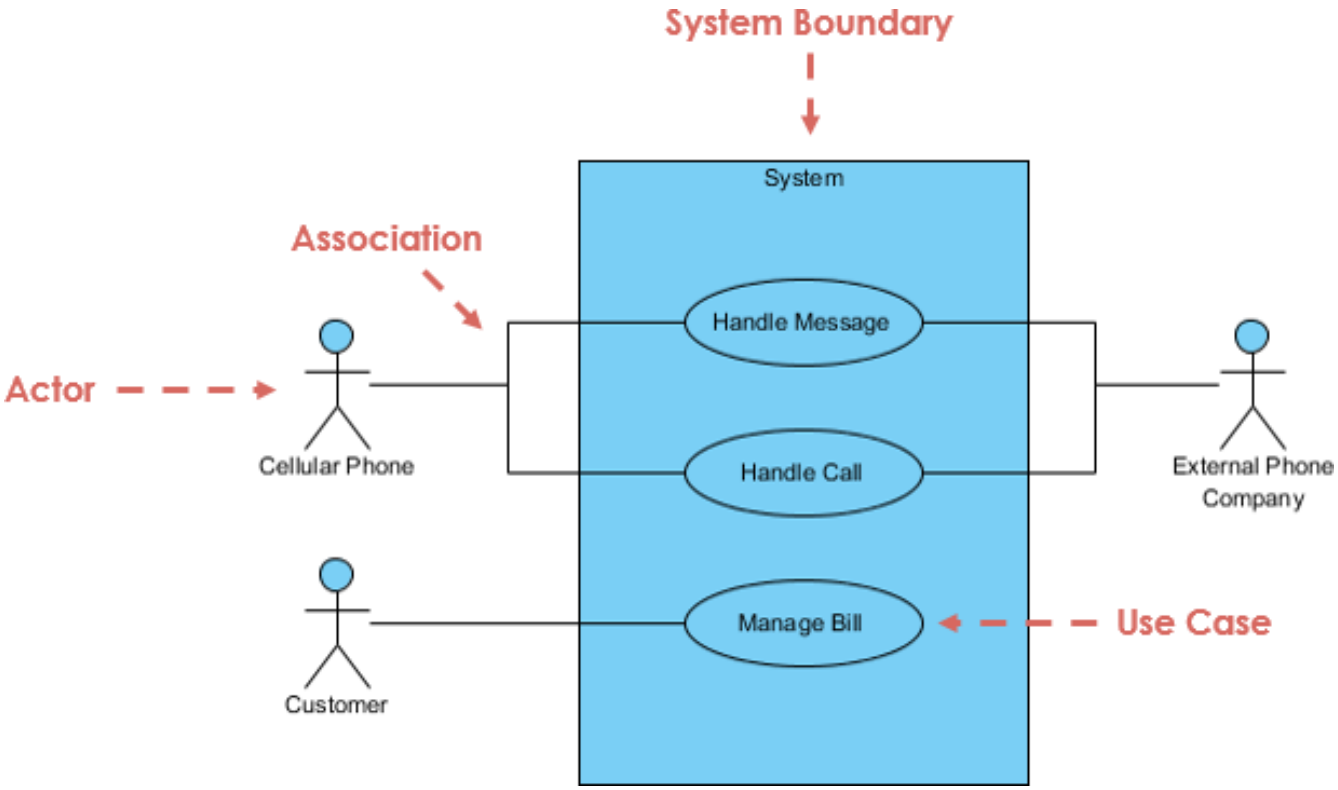
A customer walks into the pet store and fills the cart with a variety of items. When checking out, the cashier asks if the customer has a loyalty card. If so, the cashier swipes the card, authenticating the customer. If not, then the cashier offers to complete one on the spot. After the loyalty card activity, the cashier scans products using a bar code reader. As each item is scanned, the sale is totaled and the inventory is appropriately updated. Upon completion of product scanning a subtotal is computed. Then any coupons and discounts are entered. A new subtotal is computed and applicable taxes are added. A receipt is printed and the customer pays using cash, credit card, debit card, or check. All appropriate totals (sales, tax, discounts, rebates, etc.) are computed and recorded.

- Scenarios are quite useful **when the domain is novel**. User stories are, in fact, a form of scenario.

Use Cases

- Use cases are a way for more sophisticated customers and stakeholders to describe their desiderata.
- Use cases depict the interactions between the system and the environment around the system, in particular, human users and other systems.
- They can be used to model the behavior of pure software or hybrid hardware software systems.
- Use cases describe scenarios of operation of the system from the designer's (as opposed to customers') perspective.
- Use cases are typically represented using a use case diagram.

Use Cases


System Boundary
System
Association
Handle Message
Actor
Cellular Phone
Handle Call
External Phone Company
Manage Bill
Use Case
Customer

Eliciting Nonfunctional Requirements

Eliciting Nonfunctional Requirements

- Nonfunctional requirement (NFR) elicitation techniques differ from functional requirements elicitation techniques.

- NFRs are generally stated informally during the requirements analysis, are often contradictory, and difficult to enforce and validate during the software development process.

- NFRs can face issues during four key stages of software development:
  - elicitation (gathering requirements),
  - documentation (writing them down),
  - management (keeping track of them), and
  - testing (checking if they work).

Eliciting Nonfunctional Requirements

- The biggest source of problems happens at the first stage—**elicitation**. If NFRs are missed or not properly identified when gathering requirements, these gaps can cause issues throughout the entire development process.

- The reasons are, for instance, that:
  1. certain constraints are unknown at the requirements stage,
  2. NFRs tend to conflict with each other,
  3. separating FRs and NFRs makes it difficult to trace dependencies between them, whereas functional and nonfunctional considerations are difficult to separate if all requirements are mixed together.

- It is not easy to choose a method for eliciting, detailing, and documenting NERs among the variety of existing methods.

Eliciting Nonfunctional Requirements

- There are a couple of common ways to figure out non-functional requirements (NFRs):

Competitive Analysis:

- This involves comparing your system's qualities with those of competing products. For example, you can check how fast a competitor’s system responds and then decide if your system needs to be faster or better in other ways.

Pre-established Questionnaire:

- A requirements engineer can create a set of questions for stakeholders (people who have an interest in the system) and the development team.
- These questions help uncover NFRs by addressing aspects like error handling, future system changes, and data security.
- For instance, asking "How should the system handle input mistakes?" or "What data must be kept secure?" helps identify specific NFRs.

Organizing Various Elicitation Techniques Roughly by Type

Technique Type	Techniques
Domain-oriented	Card sorting, Designer as apprentice, Domain analysis, Laddering, Protocol analysis, Task analysis
Ethnography	Ethnographic observation
Goals	Goal based approaches
Group work	Brainstorming, Group work, JAD, Workshops
Interviews	Interviews, Questionnaires
Prototyping	Prototyping
Scenarios	Scenarios, Use cases, User stories
Viewpoints	Viewpoints, Repertory grids

	Interviews	Group work	Ethnography	Prototyping	Goals	Scenarios	Viewpoints
Understanding the domain	•	•	•		•	•	•
Identifying the sources of requirements	•	•			•	•	•
Analyzing the stakeholders	•	•	•	•	•	•	•
Selecting techniques and approaches	•	•					
Eliciting the requirements	•	•	•	•	•	•	•

Thank you

ICT 4203
Software Engineering

Software Specification – Part 03

Requirement Negotiation

Prasadini Padmasiri

Requirement Negotiation

Definition

- Requirements negotiation is the process where different stakeholders discuss, adjust, and agree on the project's requirements. This is vital because stakeholders often have conflicting needs, and reaching a compromise ensures a more successful project outcome.

Objective

- To align requirements with project constraints (like budget, time, and resources) while balancing different stakeholders' interests.

Importance of Requirement Negotiation

1. Aligns Stakeholder Expectations

- Diverse Needs
- Avoids Misunderstandings

2. Prevents Scope Creep

- Clear Boundaries
- Prioritization

3. Ensures Feasibility

- Technical & Resource Constraints
- Trade-offs:
  - It enables productive discussions about trade-offs. For example, if stakeholders want both high security and rapid development, negotiation helps them understand the trade-offs, such as sacrificing speed for security or adjusting the budget.

Importance of Requirement Negotiation

4. Enhances Collaboration

Builds Relationships:

- Successful requirements negotiation fosters positive relationships among stakeholders and between the development team and clients. By listening to and addressing each stakeholder's needs, you create trust and mutual respect, which is crucial for long-term collaboration.

Avoids Conflict:

- Proactively resolving conflicts during negotiation reduces the chance of disagreements escalating later in the project. If all parties feel heard and that their concerns have been addressed, they are less likely to resist decisions down the line.

Importance of Requirement Negotiation

5. Improves the Quality of the Final Product

User Satisfaction:

- Negotiating requirements ensures that the final product meets the true needs of its users. By balancing different stakeholders' priorities—such as functionality, user experience, and performance—you deliver a solution that satisfies users and customers.

Balanced Solutions:

- Instead of focusing on one aspect (e.g., only focusing on speed at the cost of usability), negotiation helps develop a well-rounded product that balances performance, security, usability, and other critical non-functional requirements.

Importance of Requirement Negotiation

6. Minimizes Risk

7. Supports Decision-Making

Stages of Requirement Negotiation

1. Requirement Gathering: At this stage, stakeholders express their needs and expectations. It's essential to record all requirements, including conflicting ones.

2. Conflict Identification: Analyze and identify where requirements clash, whether due to technical limitations, budgetary constraints, or differing priorities (e.g., usability vs. security).

3. Prioritization: Use prioritization techniques to focus on the most critical requirements. Methods like MoSCoW (Must, Should, Could, Won't have) or the Kano Model can be introduced here.

4. Negotiation & Trade-offs: Discuss and resolve conflicts by finding a balance between stakeholder needs. For example, a compromise might be made between the system's speed and its security features.

5. Documenting Decisions: Once a consensus is reached, document the agreed-upon requirements to avoid future disputes.

Negotiation Strategies

1. Win-Win Negotiation - Strive for a solution where all stakeholders feel that their key needs are met, rather than a win-lose scenario where one side compromises too much.

2. Trade-offs and Concessions - To find balance when it's not possible to meet all requirements.

3. Active Listening and Clarification - Understand the underlying needs behind stakeholders' requests.

4. Compromise - Reach a middle ground where both sides make sacrifices, but each party gains something important.

5. Mediation - When stakeholders reach an impasse, a neutral third party helps facilitate resolution.

Thank you

Agile Software Development

Amila Sandaruwan
Associate Tech Lead
Calcey Technologies

Dinesh Silva
Technical Lead Engineer
Neem

Agile Software Development

- Practiced for 15+ years in the software industry, with very rapid growth during recent past.

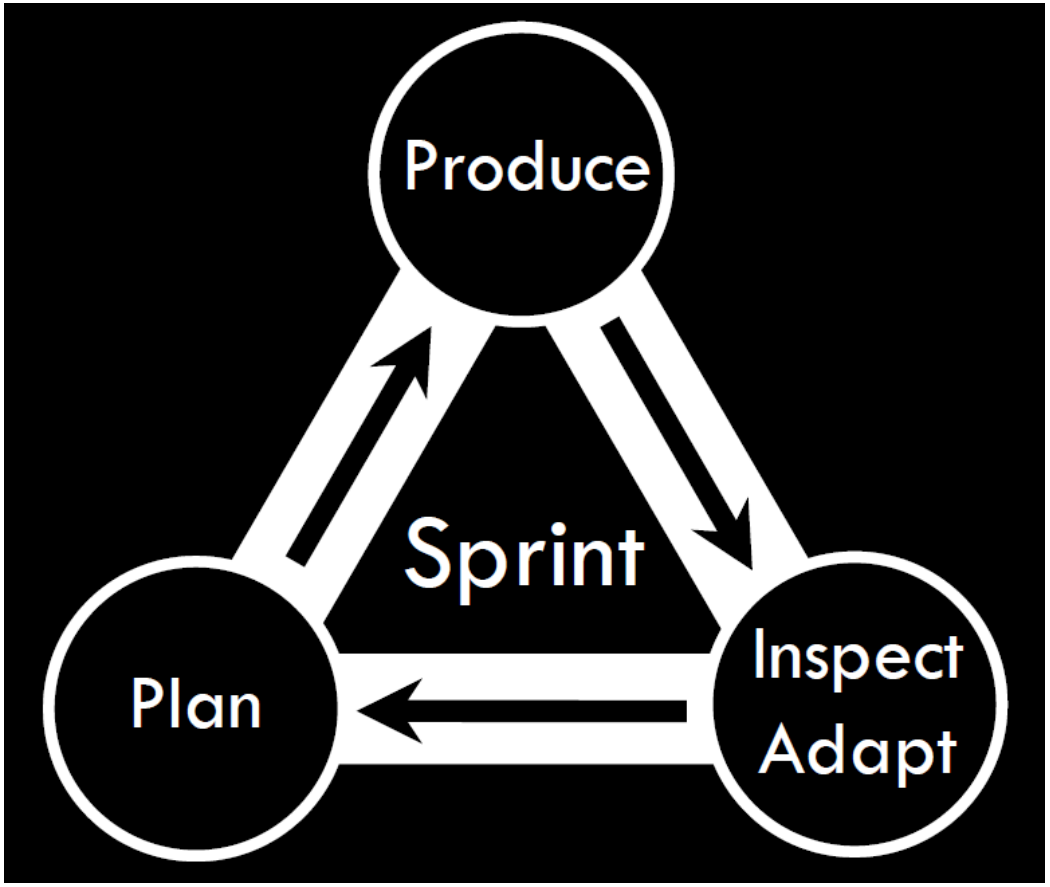
- Agile is now in use at most of the Fortune 100, such as:
    - Google
    - Oracle
    - SAP
    - GE
    - TCS
    - Intel
    - Microsoft
    - Infosys
    - HP
    - Target
    - Wipro
    - BT

The Agile Manifesto

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

While there is value in the items on the right, we value the items on the left more.

SCRUM



SCRUM Team

- There is a cross-functional Development Team –it has the diverse set of skills necessary to produce “done” product in a Sprint.
- The Dev Team is self-organizing – it figures out how to get work done.
- The Product Owner decides what product needs to be produced.
- The Dev Team decides how much product to target in each Sprint.
- The Dev Team tries to hit the target, but it can over-or under-deliver.
- Each Sprint is a time box, and is never extended.
- The Dev Team aims for “done” each Sprint = tested and defect-free.
- At the end of the Sprint, we inspect and adapt product and practices.

Development Team

- The Dev Team is responsible for implementing the product.

- Recommended size is 5-9 people. (Bigger projects will have multiple Dev Teams working in parallel with each other.)

- **Self-organizing** – Dev Team works together to set a realistic target for the Sprint, and they do their best to hit that target. They are responsible for delivering high-quality work at a sustainable pace.

- **Cross-functional** – Dev Team has all the diverse skills needed to produce working software in a Sprint. Each person might have one (or more) of the following skills: architecture, coding, testing, documentation, etc. We call all these people “Developers”.

Scrum Master

- The Scrum Master is a "servant leader", helping the Dev Team and Product Owner use Scrum to achieve the best possible results today, and improve their results in the future.

- … but not the Dev Team’s boss.

Product Owner

- Product Owner is the single person responsible for ensuring that the optimal business value is achieved.
- PO is responsible for schedule, scope, and cost.
- PO maintains the Product Backlog, a prioritized list of everything that needs to be done.
- PO role should be played by the customer –or someone who deeply understands customer needs.

Product Backlog Items

- The Product Owner gathers a list of ideas to be implemented (things like features and functionality) plus other work to be done (to mitigate risks, answer questions, etc.) from stakeholders.

- These are known as Product Backlog Items.

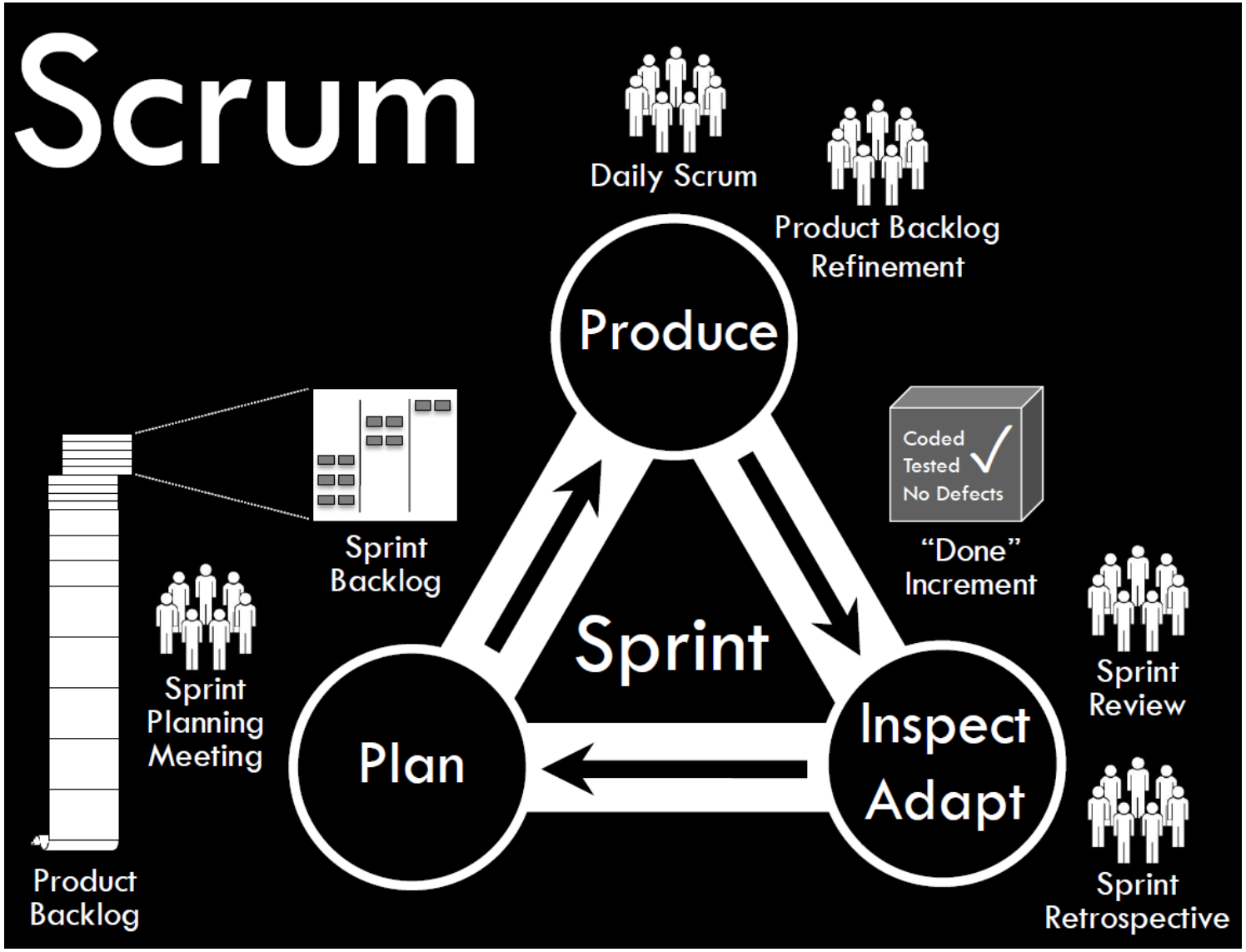
- Some teams write these as User Stories

User Stories

- User Stories are an Agile approach to writing your Product Backlog Items
- Widely used by Scrum Teams
- Not required, but highly recommend

What is a User Story?

- A User Story is a short, plain-language description of the needs of a user, centered on what they need or want to do, and why.
- As a <type of user>, I want <some goal> so that <some reason>.


Scrum
Daily Scrum
Product Backlog Refinement
Produce
Coded
Tested
No Defects
"Done" Increment
Sprint Backlog
Sprint
Sprint Planning Meeting
Plan
Inspect Adapt
Sprint Review
Sprint Retrospective
Product Backlog

Sprint Planning Meeting

- Conducted by the Scrum Team (PO, SM, Dev Team) at the start of every Sprint.
- Timeboxed to 2 hours for each week of Sprint.

Part 1: What work is to be done

- Product Owner leads the Dev Team through the highest priority Product Backlog items, so they have a clear understanding of the requirements.
- The Dev Team makes an initial estimate of how much it can complete.

Part 2: How will the work be accomplished

- The Dev Team creates a detailed plan for how they will try to deliver what they've targeted. This is the Sprint Backlog.
- The Dev Team validates the initial estimate they made.

Daily Scrum

- Purpose
  - Enable the Dev Team to give each other a brief daily update
  - Enable the Dev Team to make any blocks visible to everyone

- Daily, the Dev Team stands in a circle and reports:
  - What did I get donesince the last Daily Scrum?
  - What will I try to get doneby the next Daily Scrum?
  - What are my blocks?

- Product Owner can attend, but must not interfere
- After Daily Scrum, Scrum Master helps remove blocks, and Dev Team can meet in smaller groups

Sprint Review

- Purpose: Inspect and Adapt the Product
- Timeboxed to 1 hour per week of Sprint
- The Scrum Team and stakeholders get “hands on” with what the Dev Team has produced
    - We inspect the quality, and whether it is “done”
    - We inspect whether it truly serves customer needs
    - We try to find improvements to make in the future (Product Owner adds these on the Product Backlog)

Sprint Retrospective

- Purpose: Inspect and Adapt Our Practices
- The Scrum Team (PO, SM, Dev Team) talk about what they experienced and observed during the Sprint, both positive and negative
- They create a specific plan of action for improving their practices in the next Sprint

Thank You