Chapter 1 Summary

- Software engineering is concerned with **theories**, **methods** and **tools** for professional software development.
- Expenditure on software represents a significant fraction of **GNP** in all developed countries.
- The costs of **software** on a PC are often greater than the **hardware** cost.
- Software costs more to **maintain** than it does to **develop** (software engineering is concerned should be **cost-effective**).
- Software products:
    1) Generic products (ex: graphics programs):
        - Sold to **any customer** who wishes to buy them.
        - The specification of what the software should do and change is owned by the **software developer**.
    2) Customized products (ex: traffic monitoring systems):
        - Sold to **specific customer** to meet their own needs.
        - The specification of what the software should do and change is owned by the **customer**.
- Essential attributes of good software:

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

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
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.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
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.
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.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software economically and quickly.
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.
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 analysable specification to be developed. **You can't, therefore, say that one method is better than another**.
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.

- Engineering discipline: Using appropriate **theories and methods** to solve problems bearing in mind **organizational and financial constraints**.
- Not just technical process of development. Also, **project management and the development of tools, methods** etc. to support software production.
- For most types of system, the majority of costs are the **costs of changing the software** after it has gone into use.
- Software process activities:
  - 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**.
- General issues that affect most software:
  - Heterogeneity.
  - Business and social change.
  - Security and trust.
- The software engineering methods and tools used depend on **the type of application being developed**, **the requirements of the customer** and the **background of the development team**.
- Application types:
  - Stand-alone applications (ex: Microsoft office)
  - Interactive transaction-based applications (ex: e-commerce applications)
  - Embedded control systems (ex: traffic monitoring systems)
  - Batch processing systems.
  - Entertainment systems.
  - Systems for modelling and simulation (developed by **scientists** and **engineers** to model physical processes or situations)
  - Data collection systems (ex: collect data using a set of sensors)
  - Systems of systems.

- Some fundamental principles apply to all types of software system, systems should be developed using a managed and understood development process, dependability and performance are important, understanding and managing the software specification and requirements are important as well. Where appropriate, you should reuse software that has already been developed rather than write new software.
- Web services allow **application functionality** to be accessed over the web (or on cloud computing -users do not buy software buy pay according to use-) rather than local systems.
- Software reuse is the dominant approach for constructing web-based systems.
- Web-based systems should be developed and delivered incrementally.
- User interfaces are constrained by the capabilities of web browsers.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- <u>Issues of professional responsibility:</u>
  - Confidentiality السرية
  - Competence (عدم الكذب في مستوي مهارتهم) المهارة
  - Intellectual property rights حقوق الملكية الفكرية
  - Computer misuse
- The professional societies in the US have cooperated to produce a code of ethical practice (**ACM/IEEE Code of Ethics)**.

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Chapter 2 Summary

- **The software process:** A structured set of activities required to develop a software system.
- Process descriptions may include:
  - **Products**, which are the outcomes of a process activity;
  - **Roles**, which reflect the responsibilities of the people involved in the process;
  - **Pre- and post-conditions**, which are statements that are true before and after a process activity has been enacted or a product produced.
- **Plan-driven processes** are processes where all of the process activities are planned in advance and progress is measured against this plan.
- **Agile processes** are processes where planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- Software process models:
  - The waterfall model (plan-driven model)
  - Incremental development (plan-driven or agile)
  - Reuse-oriented software engineering (assembled from existing components and may be plan-driven or agile)
- Waterfall model phases:
  - There are separate identified phases in the waterfall model.
  - The main drawback is the **difficulty of accommodating change** after the process is underway.
  - **Inflexible partitioning of the project into distinct stages** makes it difficult to respond to changing customer requirements.
  - The waterfall model is mostly used for **large systems engineering projects** where a system is developed at several sites.

- Incremental development:
  - The cost of accommodating changing customer requirements is **reduced**.
  - It is easier to get customer **feedback** on the development work that has been done.
  - **More rapid delivery and deployment** of useful software to the customer is possible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is **not cost-effective** to produce documents that reflect every version of the system.
  - System structure tends to **degrade as new increments are added***.*
- Reuse-oriented software engineering:
  - Process stages: **Component analysis, Requirements modification, System design with reuse, Development and integration**.
  - Reuse is now the **standard approach** for building many types of business system.
- The four basic process activities of **specification, development, validation and evolution** are organized differently in different development processes. In the waterfall model, they are organized in **sequence**, whereas in incremental development they are **inter-leaved**.
- Requirements engineering process:
  - **Feasibility study:** Is it technically and financially feasible to build the system?
  - **Requirements elicitation and analysis:** What do the system stakeholders require or expect from the system?
  - **Requirements specification:** Defining the requirements in detail.
  - **Requirements validation:** Checking the validity of the requirements.
- The activities of design and implementation are closely related and may be **inter-leaved**.
- **Architectural design***,* where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- **Interface design***,* where you define the interfaces between system components.

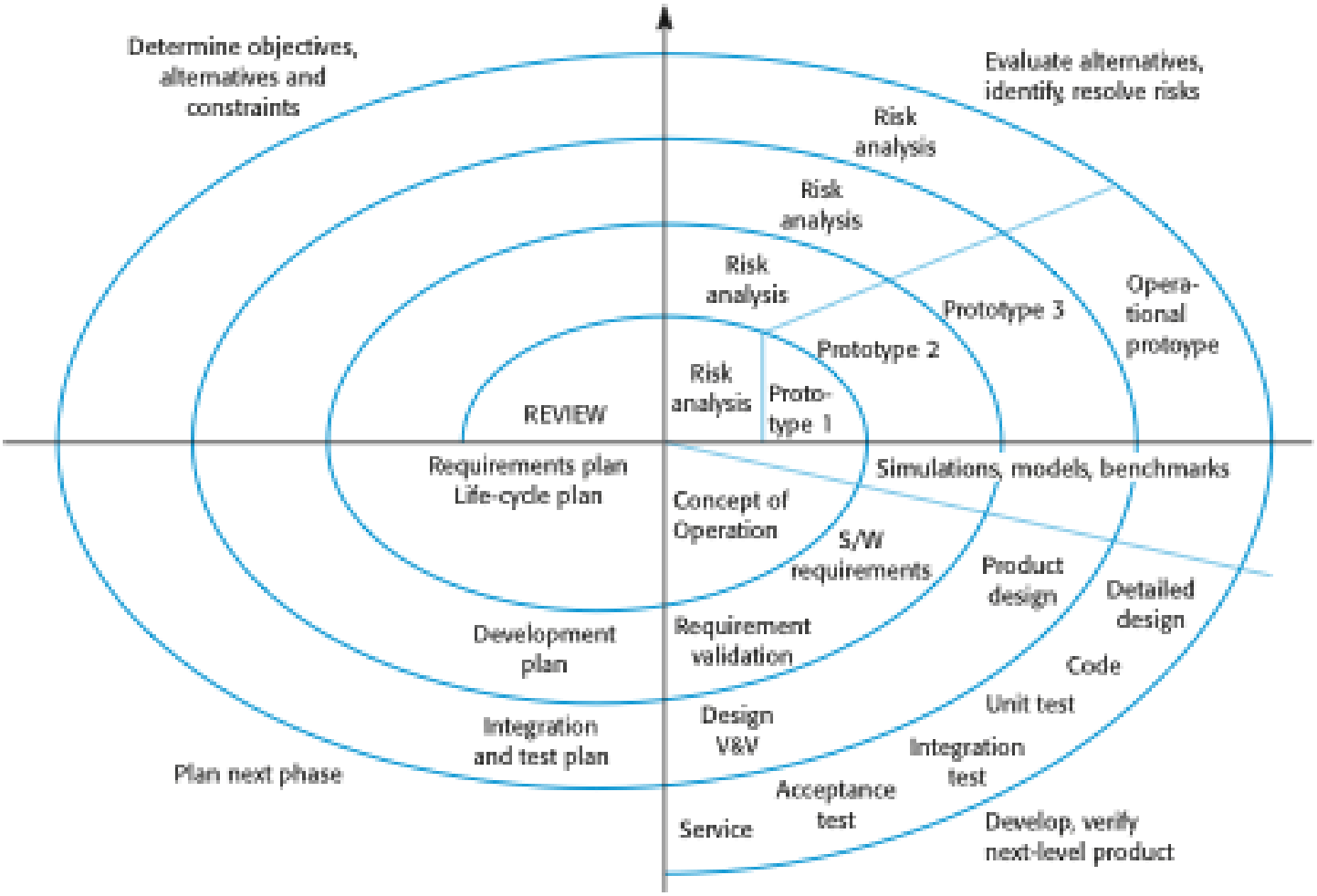
- **Component design**, where you take each system component and design how it will operate.
- **Database design**, where you design the system data structures and how these are to be represented in a database.
- **Testing** is the most commonly used **verification and validation** (V & V) activity.
- **Development or component testing:** individual components are tested independently;
- **System testing:** testing of the system as a whole.
- **Acceptance testing:** testing with customer data to check that the system meets the customer's needs.
- Software evolution takes place when you **change existing software systems** to meet new requirements as fewer and fewer systems are completely new.
- Change leads to rework so the costs of change include both **rework** (e.g. re-analyzing requirements) as well as the costs of **implementing new functionality**
- Change **avoidance**, where the software process includes activities that can anticipate possible changes before significant rework is required.
- Change **tolerance**, where the process is designed so that changes can be accommodated at relatively low cost.
- A prototype is an initial version of a system used to demonstrate concepts and try out design options, can be used in:
  - The **requirements engineering process** to help with requirements elicitation and validation;
  - In **design processes** to explore options and develop a UI design;
  - In the **testing process** to run back-to-back tests.
- Benefits of prototyping: **Improved system usability, a closer match to users' real needs, improved design quality, improved maintainability, reduced development effort.**

- Prototype development:
  - Prototype should focus on areas of the product that are **not well-understood**;
  - **Error checking and recovery** may not be included in the prototype;
  - Focus on **functional rather than non-functional requirements** such as reliability and security.
- Prototypes should be discarded after development as they are not a good basis for a production system.
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.

- Incremental development:
  - Develop the system in **increments** and **evaluate** each increment before proceeding to the development of the next increment;
  - Normal approach used in **agile methods**;
  - Evaluation done **by user/customer proxy**.
- Incremental delivery:
  - Deploy an increment for use by **end-users**;
  - **More realistic evaluation** about practical use of software;
  - Difficult to implement for replacement systems as increments have **less functionality** than the system being replaced.
- Incremental delivery advantages:
  - Customer value can be delivered with each increment so system functionality is **available earlier**.
  - Early increments act as a **prototype** to help elicit requirements for later increments.
  - **Lower risk of overall project failure**.
  - The highest priority system services tend to receive the most testing.
- Incremental delivery problems:
  - Most systems require a set of basic facilities that are used by different parts of the system.

- The essence of iterative processes is that the **specification** is developed in conjunction with the **software**.

- Boehm's spiral model (rarely used):



- Spiral model has been very influential in helping people think **about iteration in software processes** and **introducing the risk-driven approach to development**.
- **The Rational Unified Process:** A modern generic process derived from the work on the **UML** and associated **process**.
- Normally described from 3 perspectives
  - A **dynamic** perspective that shows **phases** over time;
  - A **static** perspective that shows **process** activities;
  - A **proactive** perspective that suggests **good practice**.

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	**Actors** who interact with the system are identified and **use cases** are developed to model the system requirements.
Analysis and design	A design model is created and documented using **architectural models, component models, object models and sequence models.**
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Testing	Testing is an **iterative process** that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system.
Project management	This supporting workflow manages the system development.
Environment	This workflow is concerned with **making appropriate software tools available** to the software development team.

- RUP phases:
  - **Inception:** Establish the business case for the system.
  - **Elaboration:** Develop an understanding of the problem domain and the system architecture.
  - **Construction:** System design, programming and testing.
  - **Transition:** Deploy the system in its **operating environment**.
  - **In-phase iteration:** Each phase is iterative with results developed incrementally.
  - **Cross-phase iteration:** The whole set of phases may be enacted incrementally.

- RUP good practice:
  - Develop software iteratively.
  - Manage requirements.
  - Use component-based architectures.
  - Visually model software.
  - Verify software quality.
  - Control changes to software.

Chapter 3 Summary

- **Rapid development** and **delivery** are now often the most important requirement for software systems
- Software has to evolve quickly to reflect changing business needs (without excessive rework).
- Specification, design and implementation are **inter-leaved**.
- Agile methods:
  - Focus on the **code** rather than the **design**.
  - Are based on an **iterative approach** to software development.
  - Are intended to deliver working software **quickly** and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation).
- Agile manifesto: (*while there is value in the items on the right, we value the items on the left more)*
  Individuals and interactions over **processes and tools,**
  Working software over **comprehensive documentation,**
  Customer collaboration over **contract negotiation,**
  Responding to change over **following a plan**.

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

- Agile development:
  - Product development where a software company is developing **a small or medium-sized product** for sale.
  - **Custom** system development within an organization.
  - It can be difficult to keep the interest of customers.
  - Team members may be unsuited to the intense involvement.
  - Prioritizing changes can be difficult where there are multiple stakeholders.
  - Maintaining simplicity requires extra work.
  - Contracts may be a problem as with other approaches to iterative development.
  - If agile methods are to be successful, they have to support maintenance as well as original development.
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.
- Plan-driven development (used for large systems):
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance
  - Iteration occurs within activities.

- Plan-driven approaches may be required for systems that require a lot of analysis before implementation.
- Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team (you may need to develop design documents to communicate across the development teams).
- Agile methods rely on good tools to keep track of an evolving design.
- Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code.
- You will probably be required to produce detailed documentation as part of the system safety case if the system subject to external regulation.
- Extreme programming (XP): perhaps the best-known and most widely used agile method.
- XP takes an 'extreme' approach to iterative development:
  - New versions may be built several times **per day**;
  - Increments are delivered to customers **every 2 weeks**;
  - All tests must be run for every build and the build is only accepted if tests run successfully.
- XP and agile principles:
  - Incremental development is supported through **small, frequent system releases.**
  - Customer involvement means full-time customer engagement with the team.
  - People not process through pair programming, collective ownership and a process that avoids long working hours.
  - Change supported through regular system releases.
  - Maintaining simplicity through constant refactoring of code.

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the **time available** and their relative **priority**. The developers break these stories into development 'Tasks'. These tasks are the basis of schedule and cost estimates.
Small releases	**The minimal useful set of functionalities** that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to **write tests** for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium-term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

- User requirements are expressed as **scenarios** or **user stories**.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

- Refactoring:
  - Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
  - This improves the **understandability** of the software and so reduces the need for **documentation**.
  - Changes are easier to make because the code is **well-structured and clear**.
  - However, some changes require architecture refactoring and this is much **more expensive**.
- XP has developed an approach where the program is tested after every change has been made.
- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- People adopting the customer may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.
- Test automation means that tests are written as **executable components** before the task is implemented.
- As testing is automated, there is always a set of tests that can be quickly and easily executed.
- Programmers prefer programming to testing and sometimes they take shortcuts when writing tests.
- Some tests can be very difficult to write incrementally.
- It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

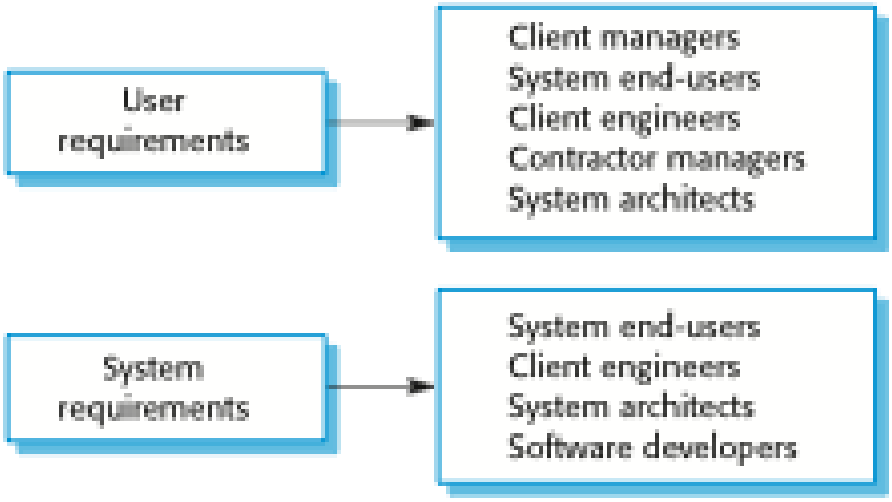
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair working together is more efficient than 2 programmers working separately.
- Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is **plan-driven**. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- The Scrum approach is a general **agile method** but its focus is on managing iterative development rather than specific agile practices.
- There are three phases in Scrum:
  - The initial phase is an outline planning phase where you establish **the general objectives for the project** and design the software architecture.
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.
- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.

- The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.
- The team then, is isolated from the customer and the organization, with all communications channelled through the so-called 'Scrum master' that protects the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders.
- The 'Scrum master' is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
- <u>Scrum benefits:</u>
  - The product is broken down into a set of manageable and understandable chunks.
  - Unstable requirements do not hold up progress.
  - The whole team have visibility of everything and consequently team communication is improved.
  - Customers see on-time delivery of increments and gain feedback on how the product works.
  - Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development.
- 'Scaling up' is concerned with using agile methods for developing large software systems that cannot be developed by a small team.
- 'Scaling out' is concerned with how agile methods can be introduced across a large organization with many years of software development experience.
- When scaling agile methods, it is essential to maintain agile fundamentals.

- For large systems development, it is not possible to focus only on the code of the system. You need to do more up-front design and system documentation.
- Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities.
- There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Chapter 4 Summary

- Requirements engineering: The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- Requirement: It may range from **a high-level abstract statement** of a service or of a system constraint to a **detailed mathematical functional** specification.
- 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**.
-

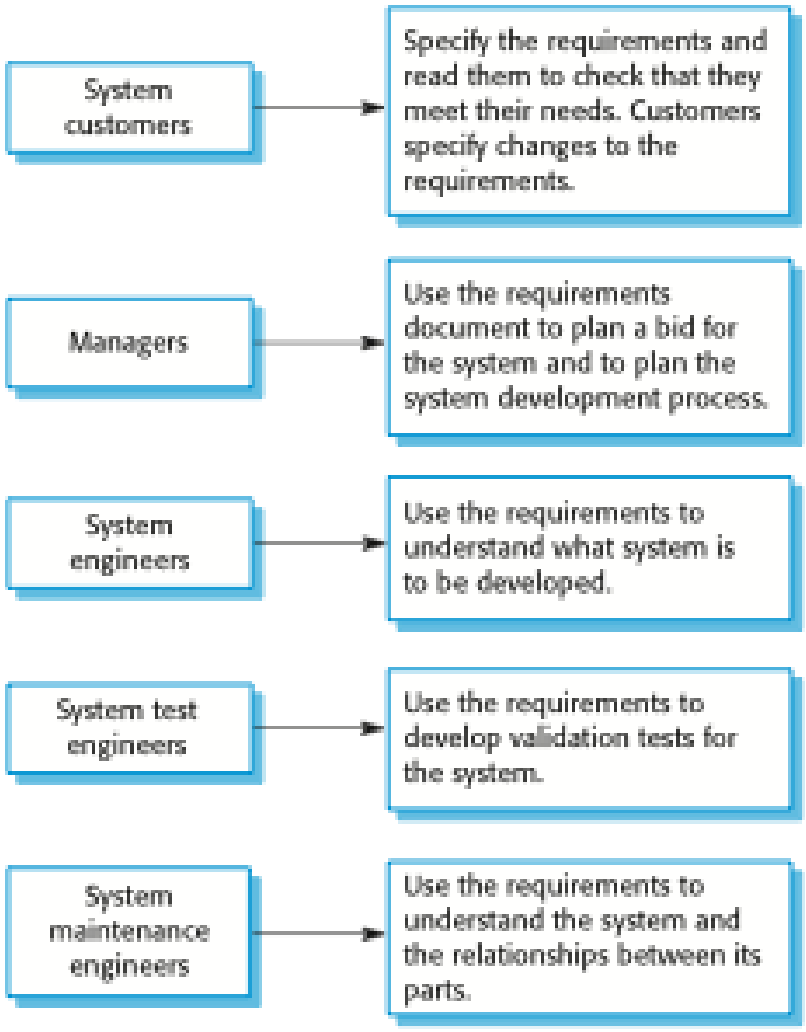


- Functional requirements:
  - Statements of **services the system should provide (in detail)**, how the system should **react to particular inputs** and how the system should **behave in particular situations**. May state what the system **should not do**.
  - Depend on the type of **software**, **expected users** and the type of system **where the software is used**.
- 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.
  - Process requirements may also be specified mandating **a particular IDE, programming language** or **development method**.
  - Non-functional requirements may be **more critical** than functional requirements. If these are not met, the system may be useless.
- Domain requirements: Constraints on the system from the **domain of operation**. Can be new **functional requirements**, constraints on existing requirements or define specific computations.
- Requirements imprecision: **Problems arise** when requirements are not precisely stated, **ambiguous requirements** may be interpreted in different ways by **developers and users**.
- In principle, requirements should be both **complete** (should include descriptions of all facilities required) and **consistent** (should be no conflicts or contradictions in the descriptions of the system facilities).
- Product requirements: Requirements which specify that the delivered product must **behave in a particular way** e.g. 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(العمل المشترك) , legislative requirements(المتطلبات التشريعية) , etc.

- Goal: A general intention of the user such as ease of use. They are helpful to developers as they convey the intentions of the system users.

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

- Verifiable non-functional requirement: A statement using some measure that can be objectively tested.
- Domain requirements problems: (1) **Understandability**: This is often not understood by **software engineers** developing the system.
  (2**)** **Implicitness**: Domain specialists understand the area so well that they do not think of making the domain requirements explicit.
- The software requirements document: Should include both a **definition of user requirements** and a **specification of the system requirements**. It should set of WHAT the system should do rather than HOW it should do it.
- Many agile methods argue that producing a requirements document is a waste of time as requirements change so quickly (The document is therefore always out of date).
- The requirements may be part of a contract for the system development



- In principle, requirements should state what the system should do and the design should describe how it does this, requirements and design are inseparable
- Problems with natural language: Lack of clarity, requirements confusion, requirements amalgamation.
- Structured specifications: An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way. (in some types of systems as embedded control system)
- Tabular specification: Particularly useful when you have to define a number of possible alternative courses of action.
- Requirements engineering processes: Requirements **elicitation**, Requirements **analysis**, Requirements **validation**, Requirements **management**.

- In practice, requirement engineering is an **iterative** activity in which these processes are **interleaved**.;
- Requirements elicitation and analysis: Involves **technical staff** working with **customers** to find out about the **application domain**, the **services** that the system should provide and the **system's operational constraints**.
- Stages include:
  - **Requirements discovery** (Interacting with range of stakeholders to discover their requirements. Domain requirements are also discovered at this stage)
  - **Requirements classification and organization** (Groups related requirements and organises them into coherent clusters)
  - **Requirements prioritization and negotiation** (Prioritising requirements and resolving requirements conflicts)
  - **Requirements specification** (Requirements are documented and input into the next round of the spiral)
- Problems of requirements analysis:
  - Stakeholders don't know what they really want.
  - Stakeholders express requirements in their own terms.
  - Different stakeholders may have conflicting requirements.
  - Organisational and political factors may influence the system requirements.
  - The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.
- Types of interview:
  - Closed interviews based on **pre-determined list of questions**.
  - Open interviews where various issues are explored with stakeholders.
- In practice, normally a **mix** of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding **domain requirements**.
- Scenarios should include: A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.

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

- Use-cases are a scenario-based technique in the UML which identify the actors in an interaction and which describe the interaction itself.
- Sequence diagrams may be used to add **detail to use-cases** by showing the sequence of event processing in the system.
- Ethnography is effective for **understanding existing processes** but cannot identify **new features** that should be added to a system.
- The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.
- Awareness of what other people are doing leads to changes in the ways in which we do things.
- Focused ethnography combines ethnography with prototyping.
- Requirements validation: Concerned with demonstrating that the requirements define the system that the customer really wants.
- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

- Requirements checking:
  - Validity: Does the system provide the functions which best support the customer's needs?
  - Consistency: Are there any requirements conflicts?
  - Completeness: Are all functions required by the customer included?
  - Realism: Can the requirements be implemented given available budget and technology?
  - Verifiability: Can the requirements be checked?
- Requirements validation techniques:
  - Requirements reviews: Systematic manual analysis of the requirements.
  - Prototyping: Using an executable model of the system to check requirements.
  - Test-case generation: Developing tests for requirements to check testability.
- Regular reviews should be held while the requirements definition is being formulated.
- Both **client** and **contractor** staff should be involved in reviews.
- Review checks:
  - Verifiability: Is the requirement realistically testable?
  - Comprehensibility: Is the requirement properly understood?
  - Traceability: Is the origin of the requirement clearly stated?
  - Adaptability: Can the requirement be changed without a large impact on other requirements?

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- The people who pay for a system and the users of that system are **rarely** the same people.
- Requirements management decisions: requirements identification, a change management process, traceability policies, tool support
-

The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems.
Glossary	This should define the technical terms used in the document.
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.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules.
System requirements specification	This should describe the functional and nonfunctional requirements in more detail.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment.
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.
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.

- Deciding if a requirement change should be accepted:
  - Change specification, problem analysis and costing, change implementation.

Chapter 5 Summary

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models, used to communicate with customers and help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
- **An external perspective**, where you model the context or environment of the system.
- **An interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- **A structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- **A behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.
- **Activity diagrams**, which show the activities involved in a process or in data processing.
- **Use case diagrams**, which show the interactions between a system and its environment.
- **Sequence diagrams**, which show interactions between actors and the system and between system components.
- **Class diagrams**, which show the object classes in the system and the associations between these classes.

- **State diagrams**, which show how the system reacts to internal and external events.
- Use of graphical models:
  - As a means of facilitating discussion about an existing or proposed system.
  - As a way of documenting an existing system.
  - As a detailed system description that can be used to generate a system implementation.
- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Architectural models show the system and its relationship with other systems.
- System boundaries are established to define what is inside and what is outside the system.
- The position of the system boundary has a profound effect on the system requirements.
- Context models simply show **the other systems in the environment**, not how the system being developed is used in that environment.
- Process models reveal how the **system being developed is used in broader business processes**.
- UML activity diagrams may be used to define **business process models**.
- **Modeling user interaction** is important as it helps to identify user requirements.
- **Modeling system-to-system interaction** highlights the communication problems that may arise.
- **Modeling component interaction** helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- **Use case diagrams** and **sequence diagrams** may be used for interaction modelling.
- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- Structural models may be **static** models, which show the structure of the system design, or **dynamic** models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.
- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- Generalization is an everyday technique that we use to manage complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes.
- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
- The lower-level classes are subclasses inherit the attributes and operations from their subclasses. These lower-level classes then add more specific attributes and operations.
- An aggregation model shows how classes that are collections are composed of other classes.