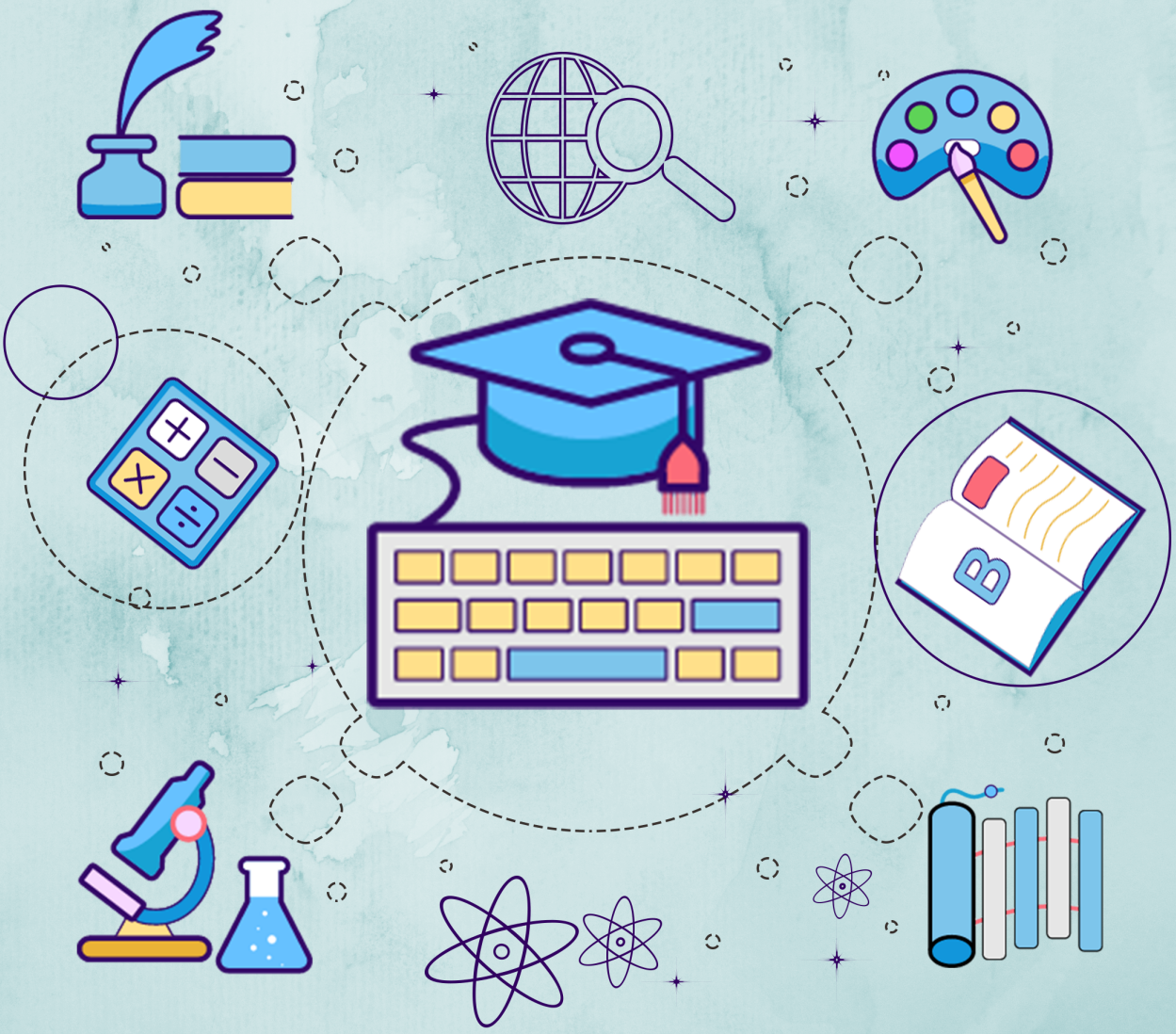
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

MANAGEMENT OF SOFTWARE SYSTEMS

CST 309

Module 1

Related Link :

- KTU S5 STUDY MATERIALS
- KTU S5 NOTES
- KTU S5 SYLLABUS
- KTU S5 TEXTBOOK PDF
- KTU S5 PREVIOUS YEAR SOLVED QUESTION PAPER

MODULE 1 : Introduction to Software Engineering (7 hours)

- 1. Introduction to Software Engineering - Professional software development, Software engineering ethics
- Software process models - The waterfall model, Incremental development. Process activities - Software specification, Software design and implementation, Software validation, Software evolution. Coping with change - Prototyping, Incremental delivery, Boehm's Spiral Model.
- Agile software development - Agile methods, agile manifesto - values and principles. Agile development techniques, Agile Project Management.
- Case studies : An insulin pump control system. Mentcare - a patient information system for mental health care.

1.1 Professional software development

- Software is not just a program themselves but also all associated documentation and configuration data .
- What is a software Engineering?

Frequently asked questions about software engineering

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.

Frequently asked questions about software engineering

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
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 analyzable 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.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
  - Organization that develops the software controls the software specification.
- Customized products(bespoke)
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.
  - Specification is developed and controlled by the organization ie buying the software.

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 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.

1.1.1 Software Engineering

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- Engineering discipline
  - Using appropriate theories and methods to solve problems within the organizational and financial constraints.
- All aspects of software production
  - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

Importance of software engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. 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
  - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- Business and social change
  - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- Security and trust
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software

1.1.2 Software Engineering diversity

- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.
- 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.

Software Engineering fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of system.
  - Understanding and managing the software specification and requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software.

Application types

- Stand-alone applications
  - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- Interactive transaction-based applications
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- Embedded control systems
  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

- Batch processing systems
  - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
- Entertainment systems
  - These are systems that are primarily for personal use and which are intended to entertain the user.
- Systems for modeling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- Data collection systems
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- Systems of systems
  - These are systems that are composed of a number of other software systems

1.1.3 Software Engineering and the Web

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Web services allow application functionality to be accessed over the web.
- Cloud computing is an approach to the provision of computer services where applications run remotely on the ‘cloud’.
  - Users do not buy software buy pay according to use

Web software Engineering

- Software reuse is the dominant approach for constructing web-based systems.
  - When building these systems, you think about how you can assemble them from pre-existing software components and systems.
- Web-based systems should be developed and delivered incrementally.
  - It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.
- User interfaces are constrained by the capabilities of web browsers.
  - Technologies such as AJAX allow rich interfaces to be created within a web browser but are still difficult to use. Web forms with local scripting are more commonly used.

1.2 Software engineering ethics

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

Issues of professional responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.
- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Rationale for the code of ethics

- *Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems.*
- *Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.*

The ACM/IEEE Code of Ethics

Software Engineering Code of Ethics and Professional Practice

- ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

- The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

- Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

Ethical principles

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.

CASE STUDIES

1.3 CASE STUDIES

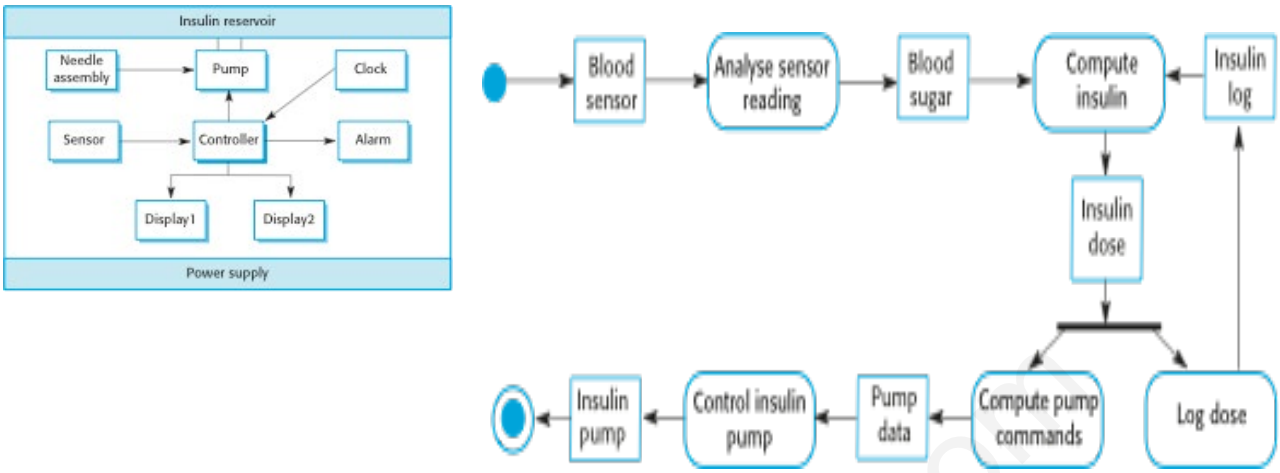
- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- A mental health case patient management system
  - A system used to maintain records of people receiving care for mental health problems.
- A wilderness weather station
  - A data collection system that collects data about weather conditions in remote areas.

1.3.1 Insulin pump control system

- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin pump hardware architecture

Activity model of the insulin pump



Essential high-level requirements

- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

1.3.2 A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

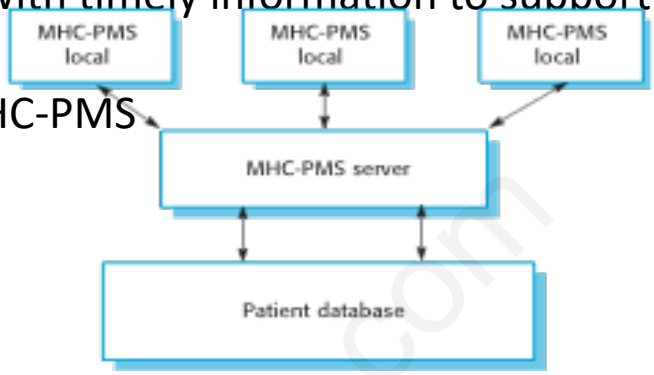
MHC-PMS

- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

MHC-PMS goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

The organization of the MHC-PMS



MHC-PMS key features

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

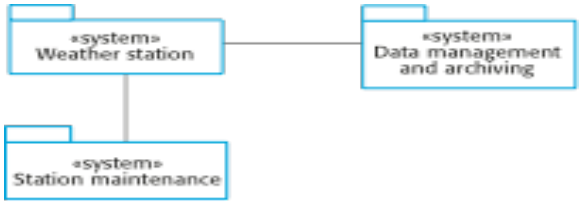
MHC-PMS concerns

- Privacy
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- Safety
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

1.3.3 Wilderness weather station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.

The weather station's environment



Weather information system

- The weather station system
  This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
- The data management and archiving system
  This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system
  This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

Additional software functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

SOFTWARE PROCESS MODELS

The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software process descriptions

- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also 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.

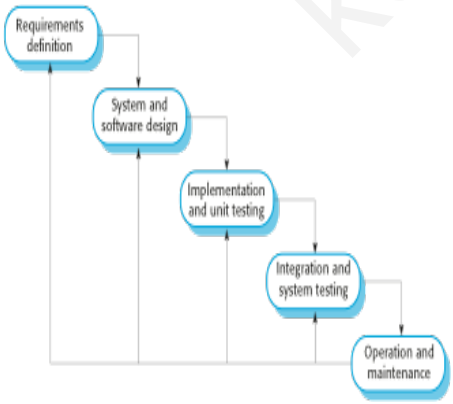
Plan-driven and agile processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, 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.
- There are no right or wrong software processes.

Software process models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Reuse-oriented software engineering
  - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



Waterfall model phases

There are separate identified phases in the waterfall model:

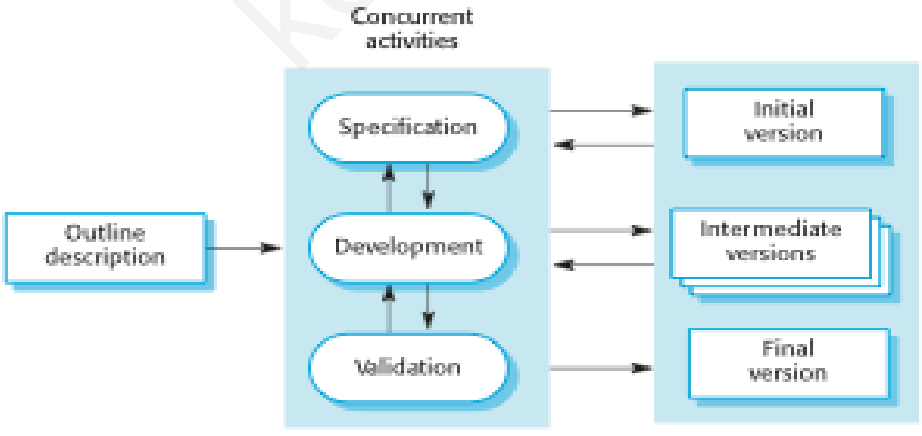
Requirements analysis and definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development

- Incremental Development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed. Specification, development and validation activities are interleaved rather than separate, with rapid feedback across activities.
- It is better for business,e-commerce and software systems.
- The customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not, then only the current increment has to be changed and possibly new functionality defined for later increments.

Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

- The process is not visible.
  - 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.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

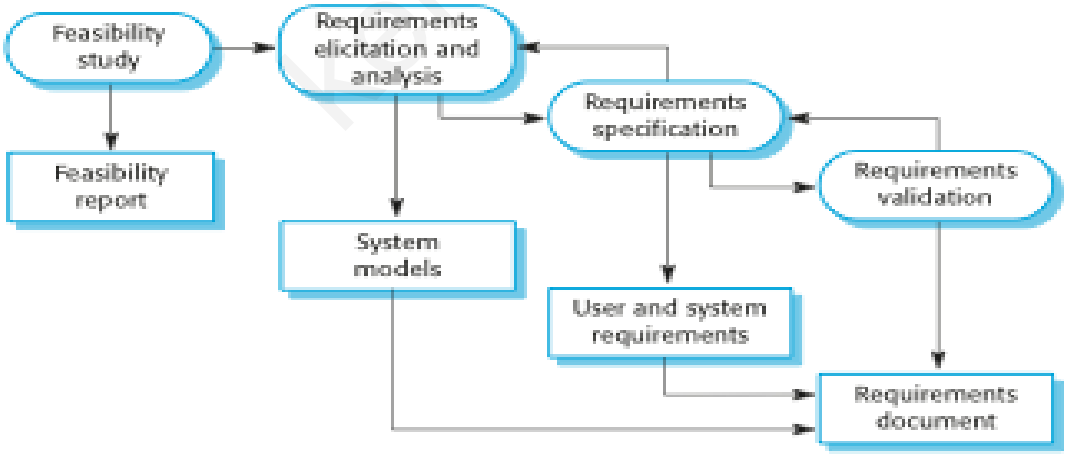
Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software 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.

Software specification/requirement Engg

- The process of establishing and defining what services are required from the system and identifying the constraints on the system's operation and development.
- Is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in system design and implementation.
- RE process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.
- Requirements are presented at two levels: End users and customers need a high level statement of the requirements; system developers need a more detailed system specification.

The requirements engineering process



Software specification

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Feasibility study
    - Is it technically and financially feasible to build the system?
    - Developed within the existing budgetary constraints.(cost effective)
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
    - Observations from existing systems, discussions with potential users ,task analysis.
    - This may involve the development of one or more models and prototypes
  - Requirements specification
    - Is the activity of translating the information gathered during the analysis activity into a document
    - Two types of requirements
    - User requirements :are abstract statements of the system requirements for the customer and end user of the system.
    - System requirements are a more detailed description of the functionality to be provided.
  - Requirements validation
    - Checking the validity of the requirements(consistent/complete)

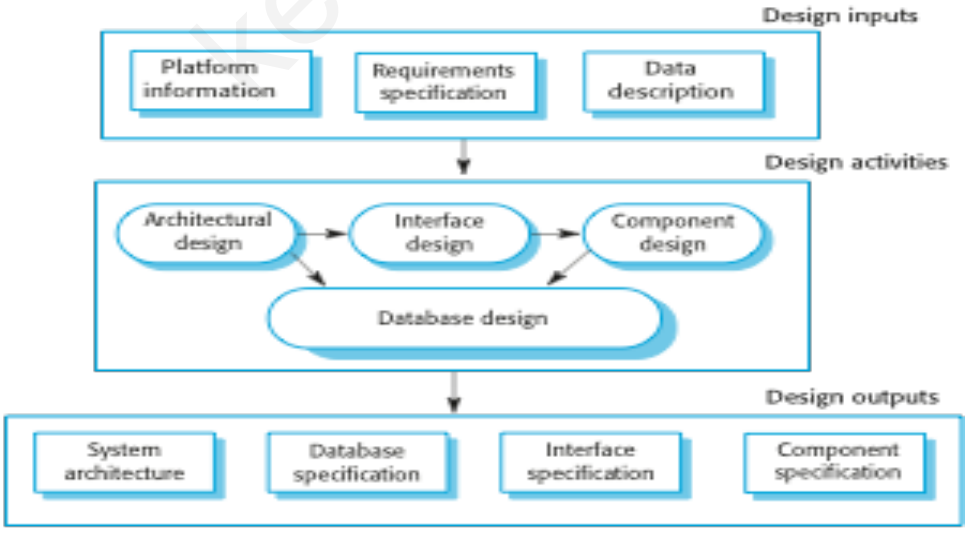
Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

Software design and implementation

- Software platform-the environment in which software will execute.
- Information about this platform is an essential input to the design process,as designers must decide how best to integrate it with the software ‘s environment.
- The requirement specification is the description of the functionality the software must provide and its performance and dependability requirements.
- If the system is to process existing data, then the description of that data may be included in the platform specification.
- Otherwise ,the data description must be an input to the design process so that the system data organization to be defined.

A general model of the design process



Design activities

- *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. This interface specification must be unambiguous.
- *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. ,depends on whether an existing database is to be reused or a new database is to be created.

- Design outputs
  - System Architecture
  - Database Specification
  - Interface specification
  - Component specification

Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Validation involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

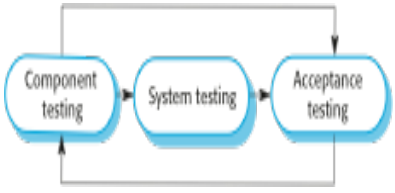
Stages of testing

.3 stage process

System components are tested(component defects are discovered early in the process)

then the integrated system is tested,(interface problems are found when the system is integrated)

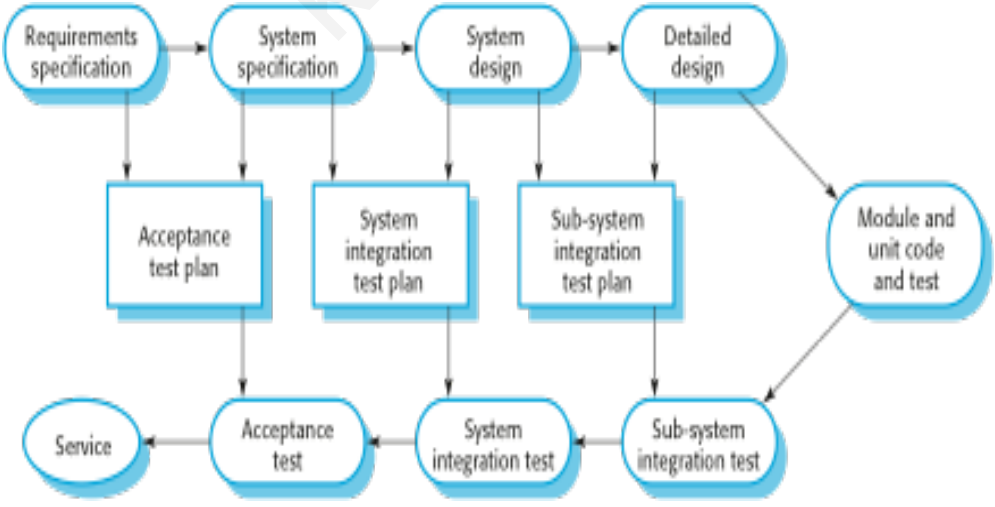
Finally the system is tested with the customers data.



Testing stages

- Development or component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
  - Test automation tools such as JUnit that can rerun component tests when new versions of the components are created, are commonly used.
- System testing
  - Testing of the system as a whole.
  - Concerned with showing the system meets its functional and non functional requirements , Testing of emergent properties is particularly important.
- Acceptance testing(alpha testing)
  - This is the final stage in the testing process before the system is accepted for operational use.
  - Testing with customer data to check that the system meets the customer's needs.

Testing phases in a plan-driven software process(v model of development)



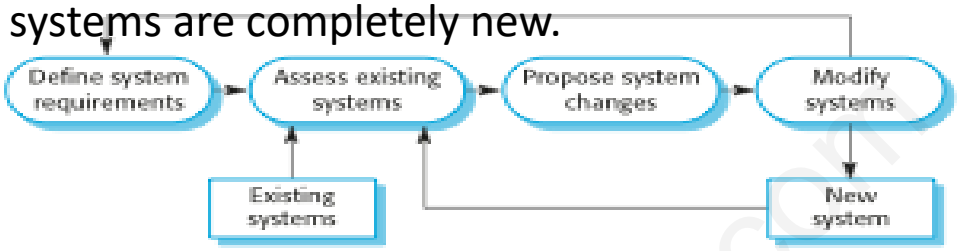
- Acceptance testing(alpha testing)
  - Alpha Testing is a **type of software testing performed to identify bugs** before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testing.
  - Custom systems are developed for a single client
  - This alpha testing process continues until the system developer and the client agree that the delivered system is an acceptable implementation of requirements.

  Beta testing
  - When software is to be marketed as a software product ,beta testing is used.
  - Beta Testing is performed by **real users** of the software application in a real environment.
  - This involves delivering a system to a number of potential users who agree to use that system.
  - They report problem to system developers.
  - This exposes the product to real use and detects errors that may not have been anticipated by the system builders.
  - After this feedback,the system is modified and released either for further beta testing or general sale.

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from beta testing will be implemented in future versions of the product.

Software evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.



Key points

- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the 'waterfall' model, incremental development, and reuse-oriented development.
- Requirements engineering is the process of developing a software specification.
- Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

Coping with change

- Change is inevitable in all large software projects.
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require application changes
- 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

Reducing the costs of rework

- Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required.
  - For example, a prototype system may be developed to show some key features of the system to customers.
- Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
  - This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Two ways of coping with change and changing system requirements:
Prototyping
Increment delivery

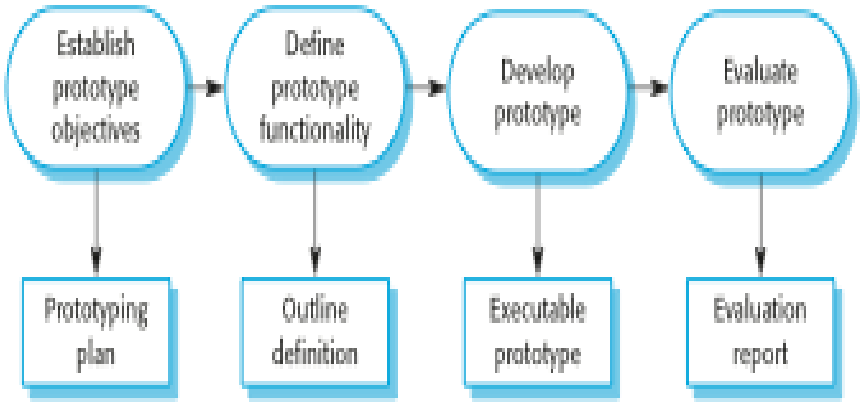
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options, and find out more about the problem and its possible solutions.
- Where a version of the system or part of the system is developed quickly to check the customer requirements .
- Rapid ,iterative development of the prototype is essential ,so that costs are controlled and system stakeholders can experiment with the prototype early in the software process.
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore particular software solutions 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.

The process of prototype development



Prototype development process

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - 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

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organisational quality standards.

Incremental development and delivery

- 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.

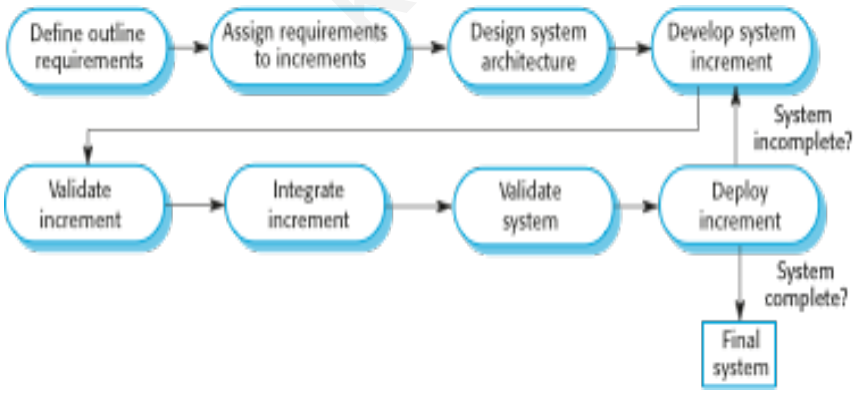
Incremental delivery

- 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 delivery

- Is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in an operational environment.
- In an incremental delivery process ,the customer identify the services to be provided by the system.
- They identify which services are most/less important.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Once an increment is completed and delivered, customers can put into service. This means that they take early delivery of part of the system functionality.
- They can experiment with the system and helps to clarify their requirements for later system increments.
- As new increments are completed ,they are integrated with the existing increments so that functionality improves with each delivered increment.

Incremental delivery



Incremental delivery advantages

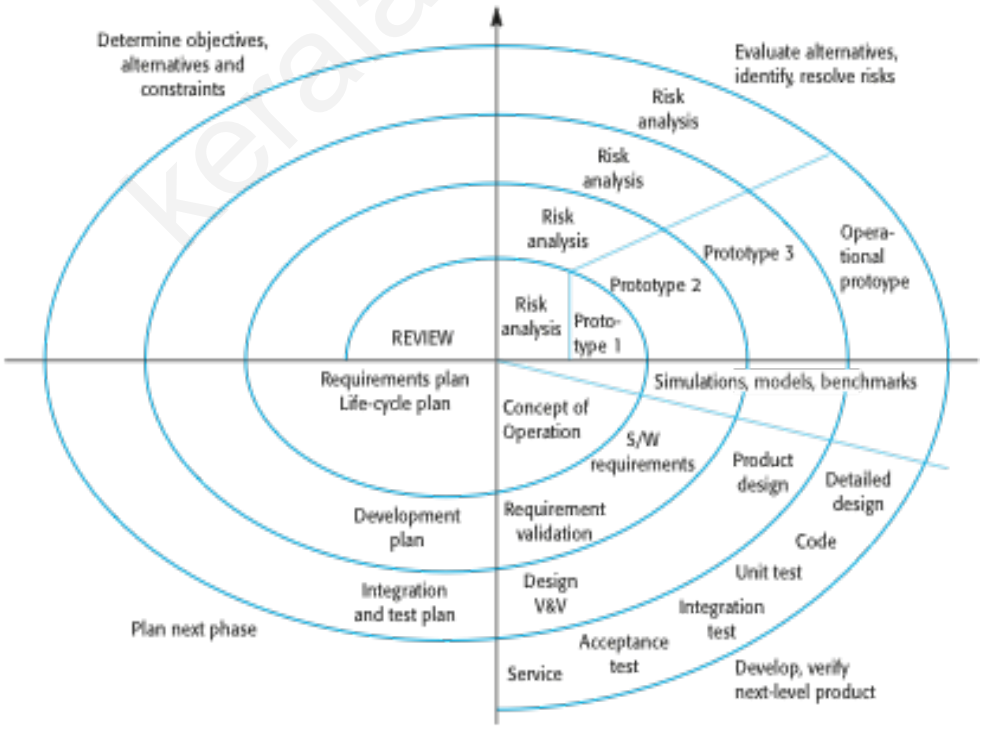
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Incremental delivery problems

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Boehm's spiral model

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- The innermost loop might be concerned with system feasibility,next is requirement definition,system design,….
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.



Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified.
  - Constraints on the process and the product are identified and a detailed management plan is drawn up.
  - Project risks are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned.

Spiral model usage

- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is rarely used as published for practical software development.

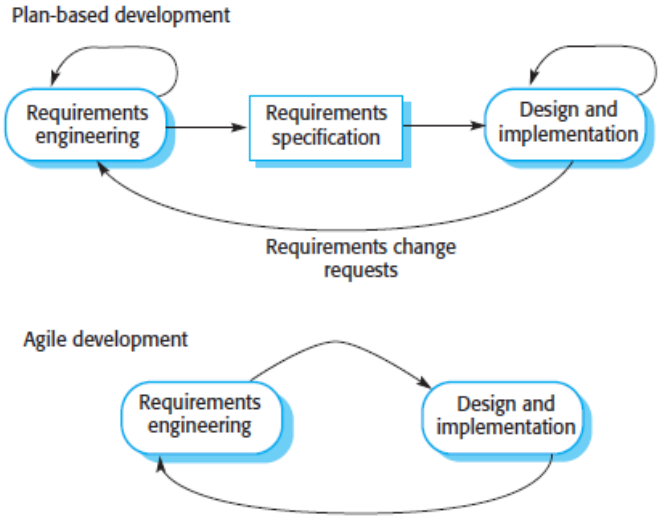
AGILE software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Agile development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - **Extensive tool support is used to support the development process**.
  - User interfaces are often developed using an IDE and graphical toolset.

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These 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) and to be able to respond quickly to changing requirements without excessive rework.

Plan driven/agile



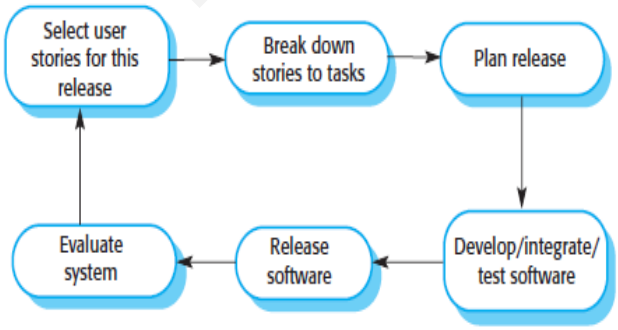
Agile manifesto

- *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
  - *Individuals and interactions over processes and tools*
    Working software over comprehensive documentation
    Customer collaboration over contract negotiation
    Responding to change over following a plan
- *That is, while there is value in the items on the right, we value the items on the left more.*

The principles of agile methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is 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 techniques:XP release cycle



In XP, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system.

Extreme programming was an agile practices that were summarized and reflect the principles of the agile manifesto:

- 1. Incremental development is supported through small, frequent releases of the system. Requirements are based on simple customer stories or scenarios that are used as a basis for deciding what functionality should be included in a system increment.
- 2. Customer involvement is supported through the continuous engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
- 3. People, not process, are supported through pair programming, collective ownership of the system code, and a sustainable development process that does not involve excessively long working hours.
- 4. Change is embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration, and continuous integration of new functionality.
- 5. Maintaining simplicity is supported by constant refactoring that improves code quality and by using simple designs that do not unnecessarily anticipate future changes to the system.

Extreme programming practices

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'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality 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.

XP practices

- **User stories:**
- Software requirements always change. In Agile methods , requirements elicitation is integrated with development by the idea of "**user stories**" where a user story is a scenario of use that might be experienced by a system user.
- After the discussion of development team with customer, they develop a "**story card**" that briefly describes a story that encapsulates the customer needs. The development team then aims to implement that scenario in a future release of the software.
- User stories may be used in planning system iterations. Once the story cards have been developed, the development team breaks these down into tasks and estimates the effort and resources required for implementing each task.
- This usually involves discussions with the customer to refine the requirements. The customer then prioritizes the stories for implementation, choosing those stories that can be used immediately to deliver useful business support.
- The intention is to identify useful functionality that can be implemented in about two weeks, when the next release of the system is made available to the customer.

- If changes are required for a system that has already been delivered, new story cards are developed and again, the customer decides whether these changes should have priority over new functionality.
- User stories can be helpful in getting users involved in suggesting requirements during an initial predevelopment requirements elicitation activity.
- **Cons:**
- The principal problem with user stories is completeness. It is difficult to judge if enough user stories have been developed to cover all of the essential requirements of a system.
- It is also difficult to judge if a single story gives a true picture of an activity. Experienced users are often so familiar with their work that they leave things out when describing it.
- **Refactoring:**
- Changes will always have to be made to the code being developed. Refactoring means that the programming team look for possible improvements to the software and implements them immediately.
- Refactoring improves the software structure and readability and avoids the structural deterioration that naturally occurs when software is changed.

Test-first development:

- Extreme Programming developed a new approach to program testing to address the difficulties of testing without a specification. Testing is automated and is central to the development process, and development cannot proceed until all tests have been successfully executed. The key features of testing in XP are:

1. test-first development:

- Write test before write the code.
- Writing tests implicitly defines both an interface and a specification of behaviour for the functionality being developed.
- Problems of requirements and interface misunderstandings are reduced.
- Test-first development requires there to be a clear relationship between system requirements and the code implementing the corresponding requirements.
- In XP, this relationship is clear because the story cards representing the requirements are broken down into tasks and the tasks are the principal unit of implementation.
- In test-first development, the task implementers have to thoroughly understand the specification so that they can write tests for the system.
- This means that ambiguities and omissions in the specification have to be clarified before implementation begins. It also avoids the problem of “test-lag.” This may happen when the developer of the system works at a faster pace than the tester.

2. Incremental test development from scenarios,

- Develop each tasks, so that the development schedule can be maintained.

3. User involvement in the test development and validation, and

- 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.

4. The use of automated testing frameworks.

- Test automation is essential for test-first development. Tests are written as executable Components before the task is implemented. These testing components should be stand-alone, should simulate the submission of input to be tested, and should check that the result meets the output specification.
- An automated test framework is a system that makes it easy to write executable tests and submit a set of tests for execution. JUnit is a widely used example of an automated testing framework for Java programs.

Pair programming:

- The programming pair sits at the same computer to develop the software. However, the same pair do not always program together. Rather, pairs are created dynamically so that all team members work with each other during the development process.

Pair programming has a number of advantages.

1. **It supports the idea of collective ownership and responsibility for the system**. This reflects Weinberg's idea of egoless programming where the software is owned by the team as a whole and individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.

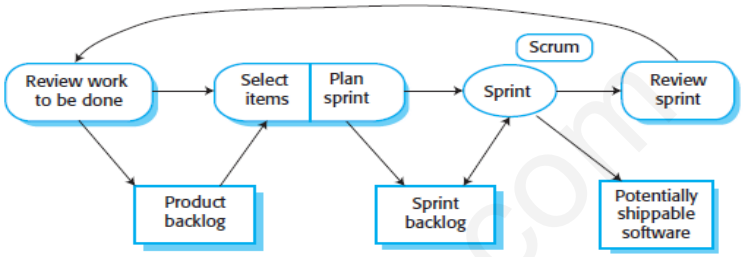
2. It acts as an informal review process because each line of code is looked at by at least

two people.

3. It encourages refactoring to improve the software structure.

Agile Project Management

- 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.
- **Scrum**
- The Scrum approach is a general agile method and focus is on managing iterative practices.
- The Scrum Process



The Sprint Cycle

- Each process iteration produces a product increment that could be delivered to customers.
- The starting point for planning is the **product backlog**, which is the list of work to be done on the project. —the list of items such as product features, requirements, user stories and engineering improvement that have to be worked on by the Scrum team.
- The product owner has a responsibility to ensure the level of specification is appropriate for the work to be done.
- Each sprint cycle lasts a fixed length of time, which is usually between 2 and 4 weeks. At the beginning of each cycle, the Product Owner prioritizes the items on the product backlog to define which are the most important items to be developed in that cycle.
- Sprints are never extended to take account of unfinished work. Items are returned to the product backlog if these cannot be completed within the allocated time for the sprint.
- The whole team is then involved in selecting which of the highest priority items they believe can be completed. They then estimate the time required to complete these items. To make these estimates, they use the velocity attained in previous sprints, that is, how much of the backlog could be covered in a single sprint. This leads to the creation of a **sprint backlog**—the work to be done during that sprint.
- The team self-organizes to decide who will work on what, and the sprint begins.

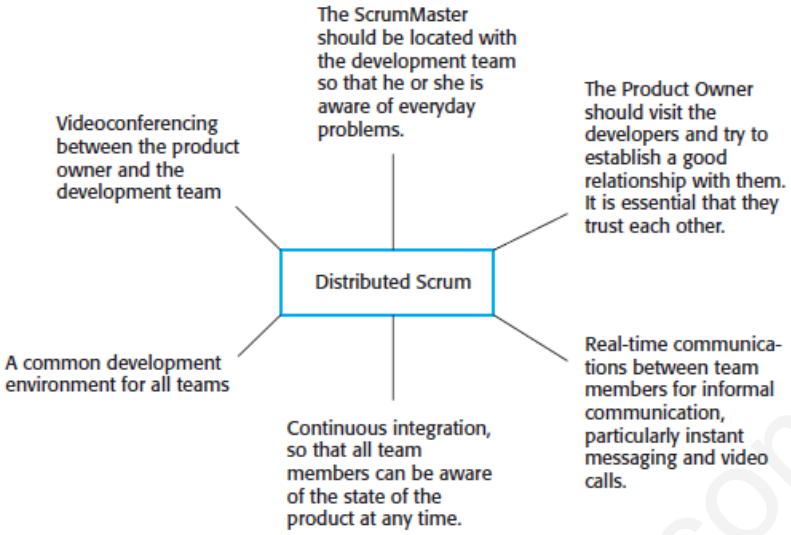
Teamwork in Scrum

- 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 (scrum)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.
- This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them, there is no top-down direction from the Scrum Master.
- Everyone participates in this short-term planning; the daily interactions among Scrum teams may be coordinated using a Scrum board. This is an office whiteboard that includes information and post-it notes about the Sprint backlog, work done, unavailability of staff, and so on. This is a shared resource for the whole team, and anyone can change or move items on the board. It means that any team member can, at a glance, see what others are doing and what work remains to be done.
- At the end of each sprint, there is a review meeting, which involves the whole team. This meeting has two purposes. First, it is a means of process improvement. The team reviews the way they have worked and reflects on how things could have been done better. Second, it provides input on the product and the product state for the product backlog review that precedes the next sprint

Scrum benefits

- 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.

- For offshore development, the product owner is in a different country from the development team, which may also be distributed. Figure shows the requirements for Distrib



Key points

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

Key points

- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation.