HNDIT4012- Software Engineering

Week 1: Introduction to Software Engineering

Topics covered

- Professional software development
  - What is meant by software engineering.
- Software engineering ethics
  - A brief introduction to ethical issues that affect software engineering.
- Case studies
  - An introduction to three examples that are used in later chapters in the book.

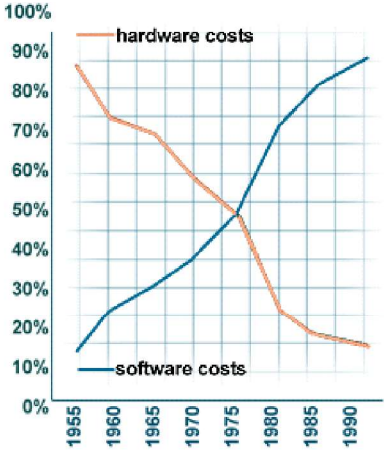
Software engineering

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
- 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.

Software costs

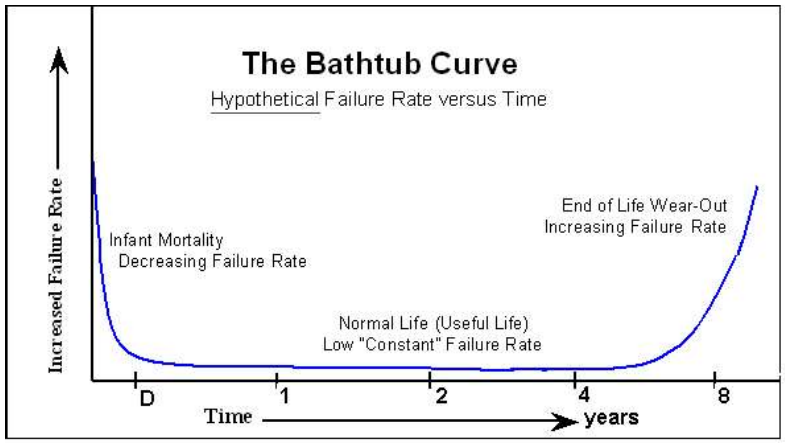
- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Hardware vs Software



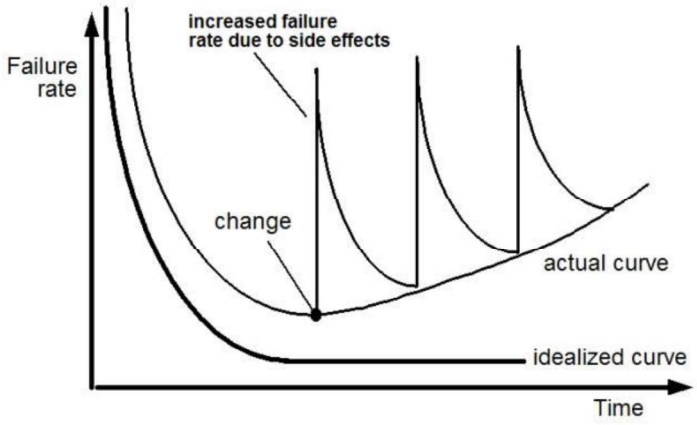
Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1

Hardware Cost



Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1

Software Failure Rate



Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1

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.
- Customized products
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1

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.

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.

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1

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.

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 bearing in mind 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.

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1

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.

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.

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.

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.

Application types

- 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 modelling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.

Application types

- 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.

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’.

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.

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.

Issues of professional responsibility

- 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

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.

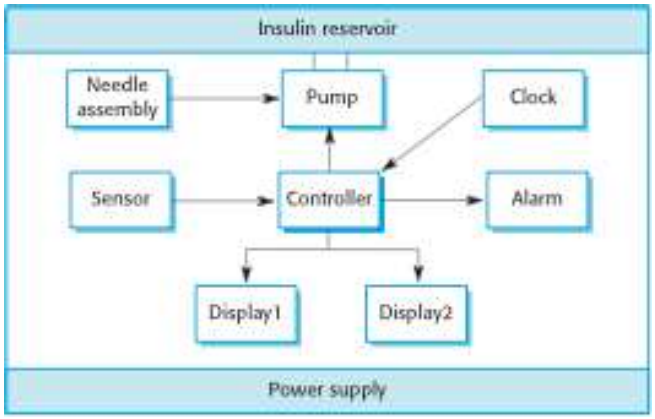
Case studies

- 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.

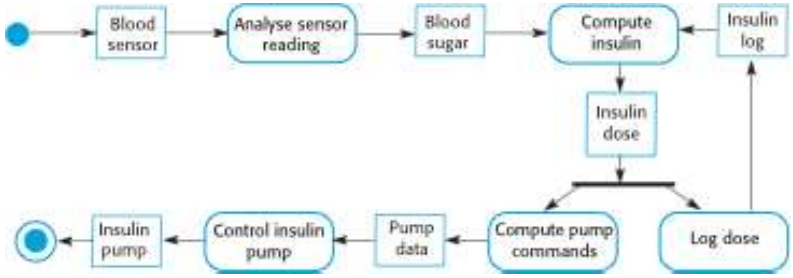
Insulin pump control system

- 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



Key points

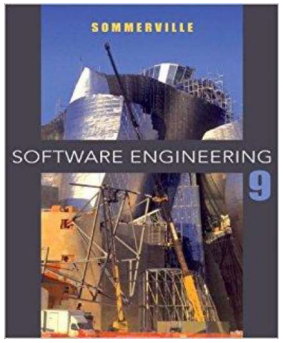
- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Essential software product attributes are maintainability, dependability and security, efficiency and acceptability.
- The high-level activities of specification, development, validation and evolution are part of all software processes.
- The fundamental notions of software engineering are universally applicable to all types of system development.

Key points

- There are many different types of system and each requires appropriate software engineering tools and techniques for their development.
- The fundamental ideas of software engineering are applicable to all types of software system.
- Professional societies publish codes of conduct which set out the standards of behaviour expected of their members.

Reference

Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 1,Chapter 2.



HNDIT 4012- Software Engineering

Week 2: Software Process

Topics covered

- Software process
- Process flow
- 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.

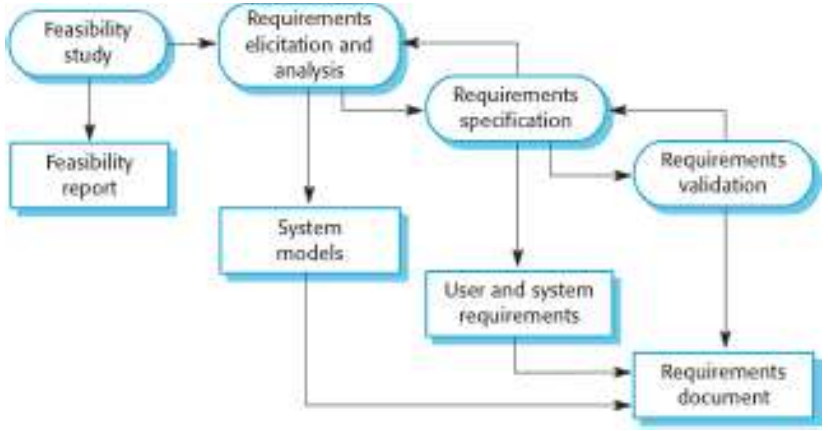
Plan-driven and agile processes

- 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.

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?
  - 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 requirements engineering process



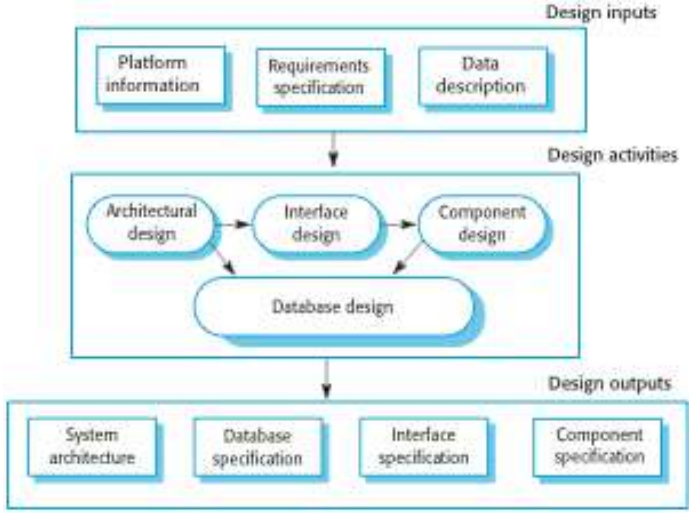
Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 2

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.

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 2

A general model of the design process



Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 2

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.
- *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.

Software validation

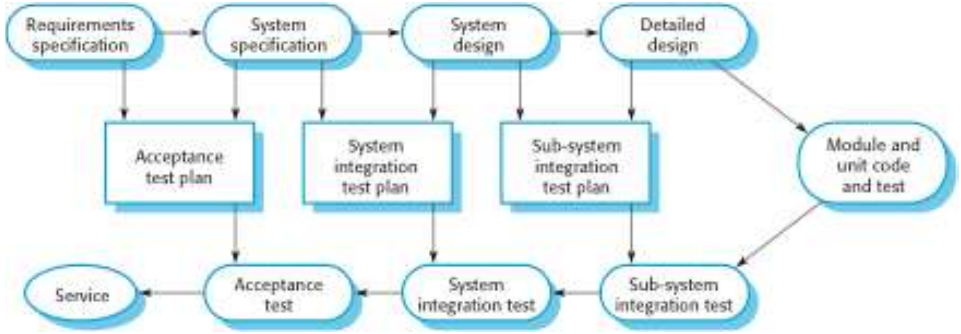
- Validation is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Stages of testing

Testing stages

- Development or component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Acceptance testing
  - Testing with customer data to check that the system meets the customer's needs.

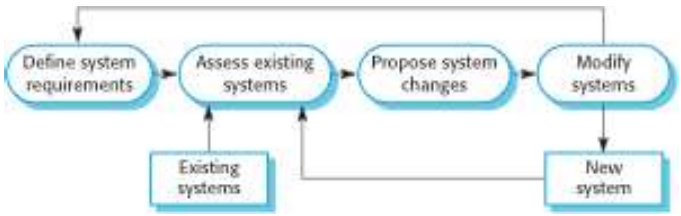
Testing phases in a plan-driven software process



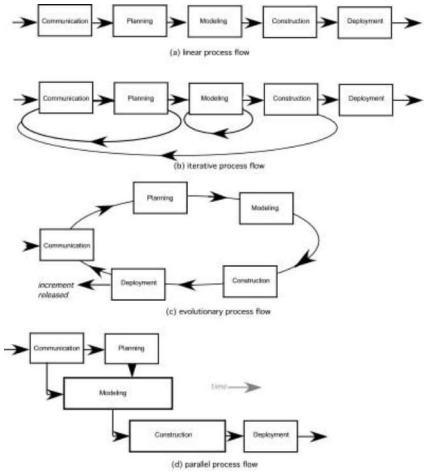
Software evolution

- 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.

System evolution



Types of Process Flow



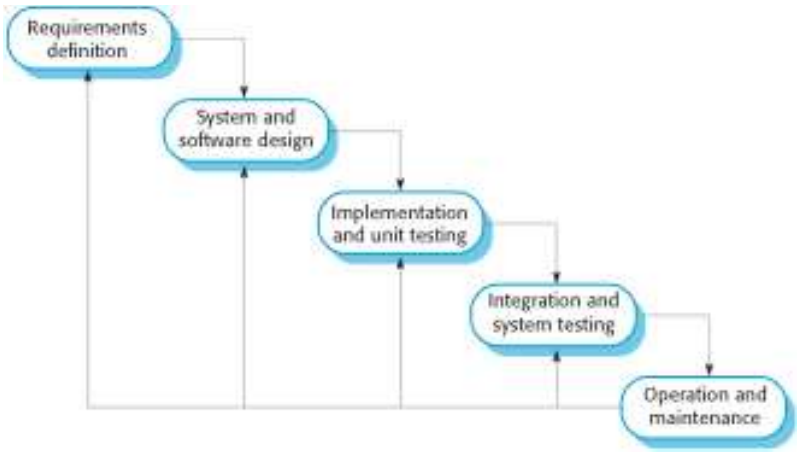
Software process models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development(Evolutionary)
  - 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

The waterfall model

- 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

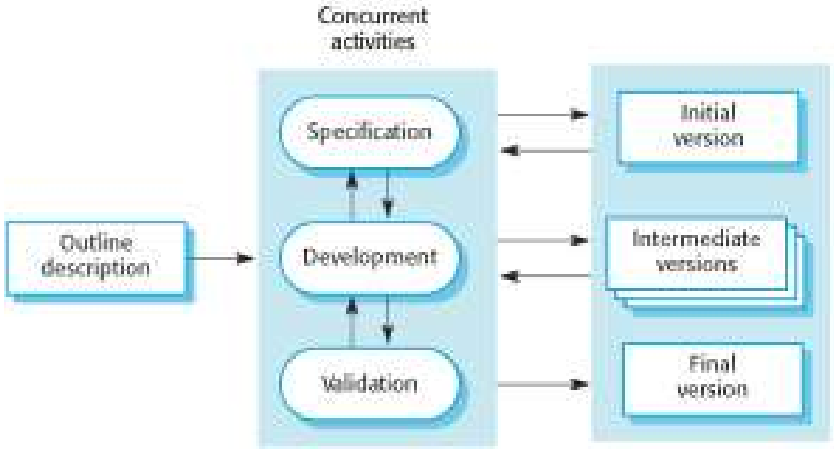
The waterfall model



The waterfall model

- The main drawback of the waterfall model 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.
  - 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.

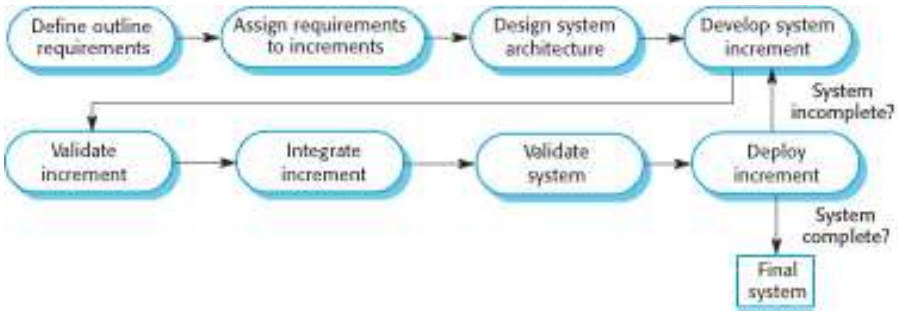
Incremental development(Evolutionary)



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.
- 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.

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.

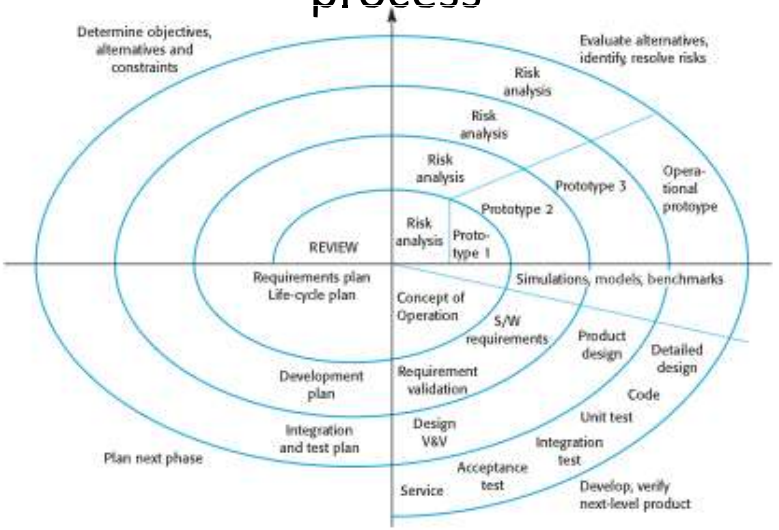
Incremental Development Types

- Type 1: Exploratory Development
  - Spiral Model
- Type 2: Throwaway Prototyping
  - Rapid Prototyping

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.
- 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.

Boehm's spiral model of the software process



Spiral model sectors

- Objective setting
  - Specific objectives for the phase 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.

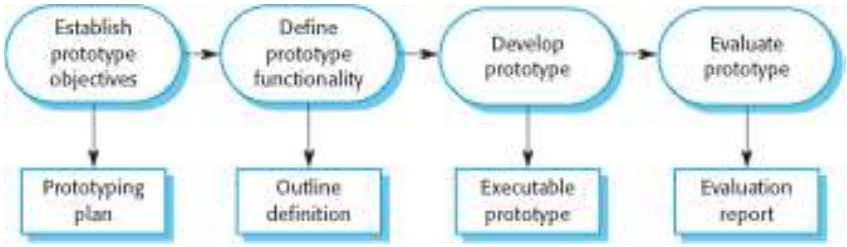
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype 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.

The process of prototype development



Prototype development

- 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

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 2

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.

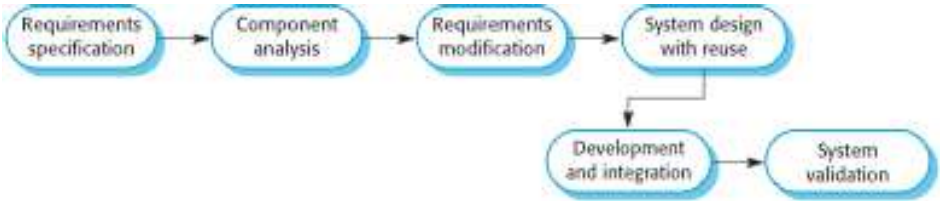
Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 2

Reuse-oriented software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- 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

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 2

Reuse-oriented software engineering



Types of software component

- Web services that are developed according to service standards and which are available for remote invocation.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Stand-alone software systems (COTS) that are configured for use in a particular environment

Key points

- 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.

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.

HNDIT 4012 Software Engineering

Week 3: Agile Software Development

Topics covered

- Adaptive Software Development
- Agile Process
- Extreme Programming
- SCRUM
- Kanban
- Lean Software Development

Agile Software Development

- Agile software development is an approach to software development that emphasizes **iterative and incremental development**, where requirements and solutions evolve through **collaboration between self-organizing**, **cross-functional teams**. It prioritizes adaptability and flexibility, allowing teams to respond to changes in requirements and feedback from users or stakeholders quickly.

Key Principles of Agile

- **Iterative Development**: The project is broken down into small increments, typically called "sprints" or "iterations," with each iteration producing a potentially shippable product increment.
- **Collaborative Approach**: Agile promotes collaboration among team members, stakeholders, and customers throughout the development process. Communication is valued over documentation, and face-to-face interactions are preferred.

Key Principles of Agile

- **Adaptive Planning**: Agile projects embrace change, allowing for adjustments to be made to the project scope, requirements, and solutions as necessary. This is in contrast to traditional "waterfall" methodologies, which often involve extensive planning upfront.
- **Continuous Feedback**: Regular feedback loops are essential in agile development. This feedback can come from stakeholders, end-users, or testing processes, helping the team to continuously improve and refine the product.

Key Principles of Agile

- **Self-organizing Teams**: Agile teams are typically self-organizing and cross-functional, meaning they have all the skills necessary to complete the work within the team. This structure promotes autonomy and empowers team members to make decisions collectively.
- **Customer Collaboration**: Agile methodologies prioritize customer collaboration, aiming to deliver value to the customer early and frequently. Customer feedback drives the direction of the development process, ensuring that the product meets the needs of its users.

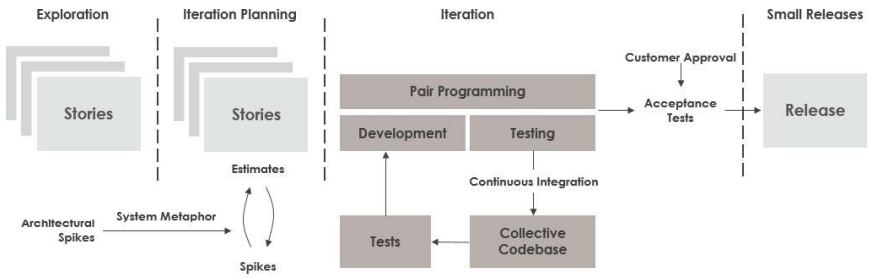
Agile Frameworks

- Popular frameworks and methodologies within the agile approach include
  - Scrum
  - Kanban
  - Extreme Programming (XP)
  - Lean Software Development.

Extreme Programming

- Extreme Programming (XP) is a software development methodology that focuses on delivering high-quality software quickly and efficiently.
- XP was created by Kent Beck in the late 1990s and has since become one of the most well-known agile methodologies.

XP Process



Scrum

- Scrum is an agile framework for managing and organizing work on complex projects, primarily used in software development but applicable to various other fields as well. It was developed in the early 1990s by Jeff Sutherland and Ken Schwaber.

Scrum Key Elements

- **Roles**: Scrum defines three primary roles:
- **Product Owner**: Represents the stakeholders and is responsible for prioritizing the product backlog, ensuring that the team is working on the most valuable features.
- **Scrum Master**: Facilitates the Scrum process, removes obstacles that hinder the team's progress, and ensures adherence to Scrum principles and practices.
- **Development Team**: Self-organizing and cross-functional, responsible for delivering the product increment during each sprint.

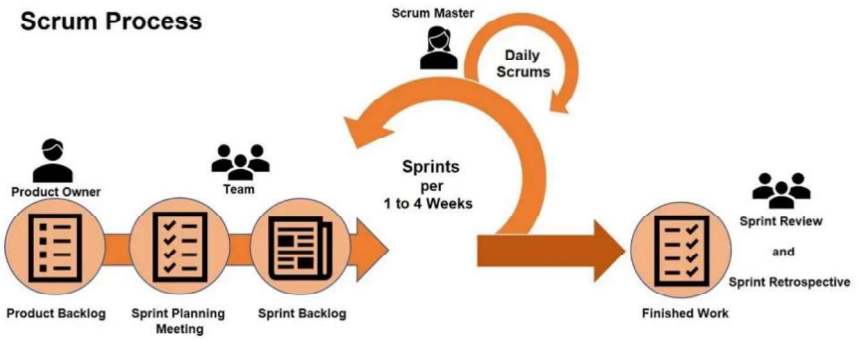
Scrum Key Elements

- **Artifacts**:
- **Product Backlog**: A prioritized list of all desired features, enhancements, and fixes for a product. It is maintained and prioritized by the Product Owner.
- **Sprint Backlog**: A subset of the Product Backlog items selected for implementation during a sprint. It is created by the Development Team during Sprint Planning.
- **Increment**: The sum of all the completed Product Backlog items at the end of a sprint. It should be in a potentially shippable state.

Scrum Key Elements

- **Events**:
- **Sprint**: A time-boxed period (usually 2-4 weeks) during which the Development Team works to deliver a potentially shippable product increment.
- **Sprint Planning**: A meeting at the beginning of each sprint where the Development Team plans the work to be done and selects the items from the Product Backlog to include in the Sprint Backlog.
- **Daily Scrum**: A short (15 minutes or less) daily meeting for the Development Team to synchronize activities, discuss progress, and identify any impediments.
- **Sprint Review**: A meeting at the end of each sprint where the Development Team demonstrates the completed work to stakeholders and gathers feedback.

Scrum Process



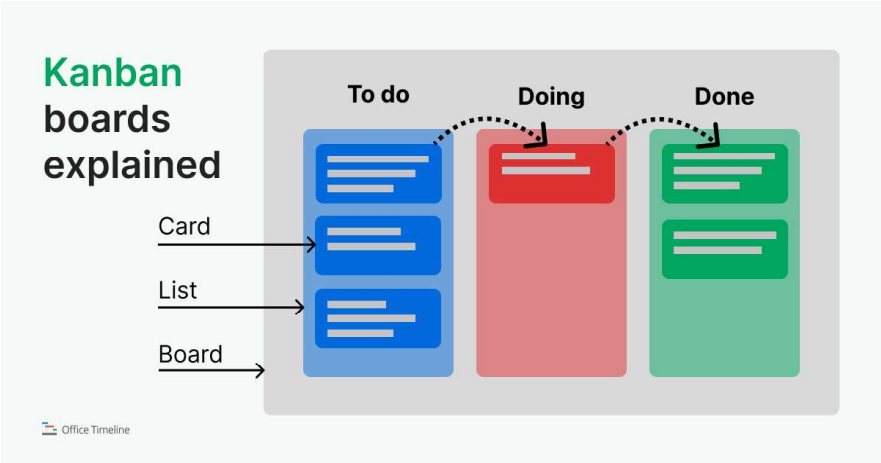
Kanban

- Kanban is a method for managing workflow, originally developed by Toyota in the 1940s as part of its manufacturing process. In recent years, it has been widely adopted in software development and various other industries to visualize and optimize work processes.

Key Elements of Kanban

- Visualization
- Limiting Work in Progress (WIP)
- Flow
- Pull-Based System
- Continuous Improvement
- Feedback Loops

Kanban Board



Lean Software Development

- Lean software development is an agile methodology inspired by the principles of lean manufacturing, originating from Toyota's production system. It focuses on maximizing customer value while minimizing waste and optimizing efficiency throughout the software development process.

Key Principles of LEAN

Discussion

- Find the following agile development methods
  - Dynamic Systems Development Method (DSDM)
  - Crystal
  - Feature Driven Development (FDD)

HNDIT 4012 Software Engineering

Week 4:Requirements Engineering

Topics covered

- Functional and non-functional requirements
- The software requirements document
- Requirements specification
- Requirements engineering processes
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

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.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process

What is a requirement?

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

User Requirement Definition

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

System Requirements Specification

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

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Functional requirements

- Example: F1 – Find Books
  - Input: the name of an author.
  - Output: details of any books by the author ,title, publisher, ISBN, etc.
  - Processing: search library catalogue for books by the specified author.

Functional Requirements-ATM System

- Check Account Balance : Write as a Functional Requirement.

Function Name: Check Account Balance

Input: Account No, PIN

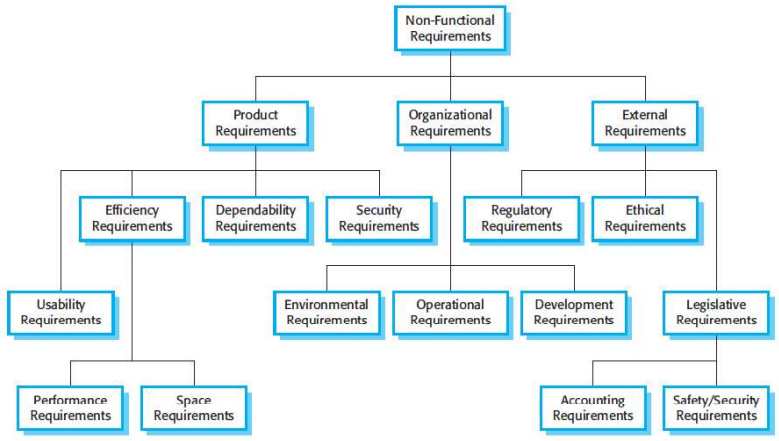
Process: Check the balance of the relevant account number and display it.

Output: Display Account Balance

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular 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.

Types of nonfunctional requirement



Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Domain requirements

- The system's operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

The software Requirements document(SRS)

- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

SRS Document

- Introduction:
  - Background, system environment
- Functional Requirements:
  - Numbered list of functions with inputs & outputs.
- Non-Functional Requirements
  - System characteristics
  - External interfaces(e.g. user interface)
- Constraints.
- Verification(Acceptance)Criteria.

Requirements specification

- The process of writing down the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.

Ways of writing a system requirements specification

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

Natural language specification-Example

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. (*Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.*)

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (*A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.*)

Structured specifications-Example

Insulin Pump/Control Software/SRS/3.3.2

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Tabular Specification-Decision Table

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 − r1) < (r1 − r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 − r1) ≥ (r1 − r0))	CompDose = round ((r2 − r1)/4) If rounded result = 0 then CompDose = MinimumDose

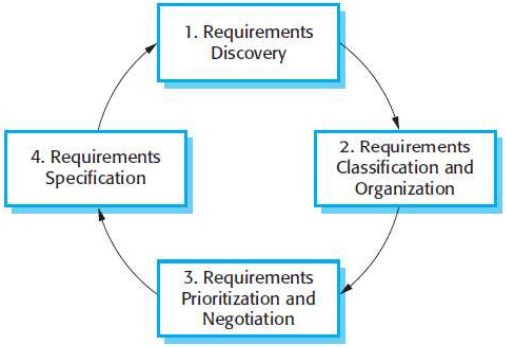
Requirements engineering processes

- There are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.

Requirements elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- 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.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

The requirements elicitation and analysis process



Problems of requirements elicitation

- 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 change.

Stakeholders

- Stakeholders range from end-users of a system through managers to external stakeholders
- For example, system stakeholders for the Hospital case study
  - Doctors
  - Patients
  - Nurses
  - Medical receptionists

Requirements discovery

- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Interaction is with system stakeholders from managers to external regulators.
- Systems normally have a range of stakeholders.

Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
  - Closed interviews based on pre-determined list of questions
  - Open interviews where various issues are explored with stakeholders.
- Effective interviewing
  - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
  - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Scenarios

- Scenarios are real-life examples of how a system can be used.
- They 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.

Scenario-Example

INITIAL ASSUMPTION:
The patient has seen a medical receptionist who has created a record in the system and collected the patient's personal information (name, address, age, etc.). A nurse is logged on to the system and is collecting medical history.

NORMAL:
The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name (first name in English) and date of birth are used to identify the patient.

The nurse chooses the menu option to add medical history.

The nurse then follows a series of prompts from the system to enter information about consultations elsewhere on mental health problems (free text input), existing medical conditions (nurse selects conditions from menu), medication currently taken (selected from menu), allergies (free text), and home life (form).

WHAT CAN GO WRONG:
The patient's record does not exist or cannot be found. The nurse should create a new record and record personal information.

Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

Patient cannot/will not provide information on medical history. The nurse should enter free text recording the patient's inability/unwillingness to provide information. The system should print the standard exclusion form stating that the lack of information may mean that treatment will be limited or delayed. This should be signed and handed to the patient.

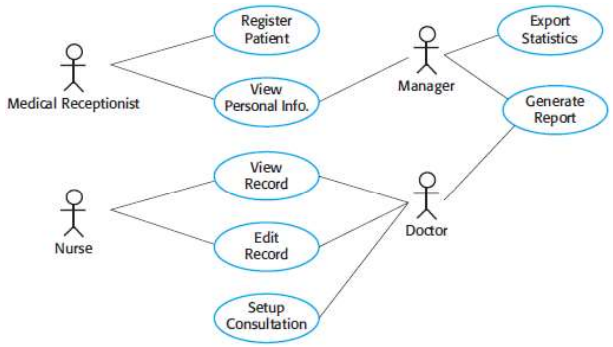
OTHER ACTIVITIES:
Record may be consulted but not edited by other staff while information is being entered.

SYSTEM STATE ON COMPLETION:
User is logged on. The patient record including medical history is entered in the database, a record is added to the system log showing the start and end time of the session and the nurse involved.

Use cases

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description (see Chapter 5).
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

Use Case-Example



Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - 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. Covered in Chapter 2.
- Test-case generation
  - Developing tests for requirements to check testability.

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

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

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.

Key points

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.

Extracted from Ian Sommerville 2011, Software Engineering, 9th edition. Chapter 4

HNDIT 4012 Software Engineering

Software Analysis & Tools

Topics covered

- DFD
- Structured Charts
- Pseudo-Code
- Decision Tables
- ERD
- Data Dictionary

Software Analysis & Design Tools

- Software analysis and design includes all activities, which help the transformation of requirement specification into implementation.
- Let us see few analysis and design tools used by software designers:

Data Flow Diagram

- Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data.
- The DFD does not mention anything about how data flows through the system.

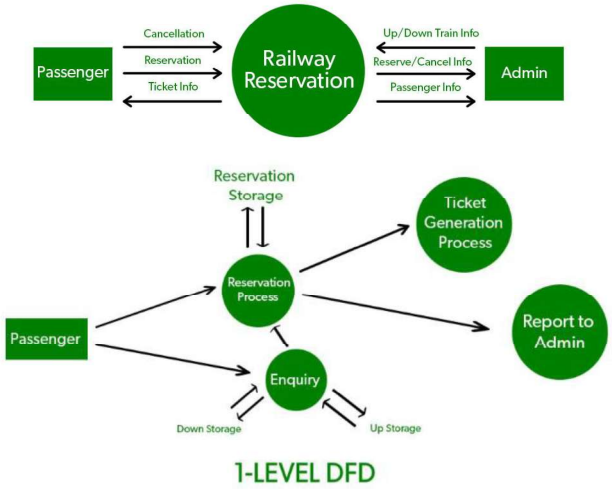
DFD Components

- **Entities** - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details.
- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD.
- **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level 1.

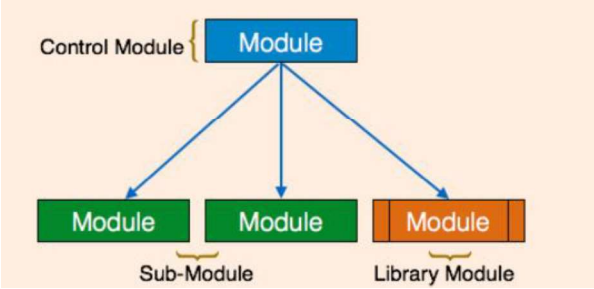
Example- DFD



Structure Charts

- Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD.
- Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Example- Structure Charts



Pseudo-Code

- Pseudo code is written more close to programming language. It may be considered as augmented programming language, full of comments and descriptions.
- Pseudo code avoids variable declaration but they are written using some actual programming language's constructs, like C, Fortran, Pascal etc.

Example-Pseudo-Code

- Change a numeric grade to a letter grade using the following rules:

Grade A: score ≥ 90

Grade B: 90 > score ≥ 80

Grade C: otherwise

Algorithm Grade

Input: a numeric score S
Output: a letter grade
1. **If** S ≥ 90 **then**
2. **Return** grade A
3. **Endif**
4. **If** S ≥ 80 and S < 90 **then**
5. **Return** grade B
6. **Else**
7. **Return** grade C
8. **Endif**

Decision Tables

- A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.
- It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

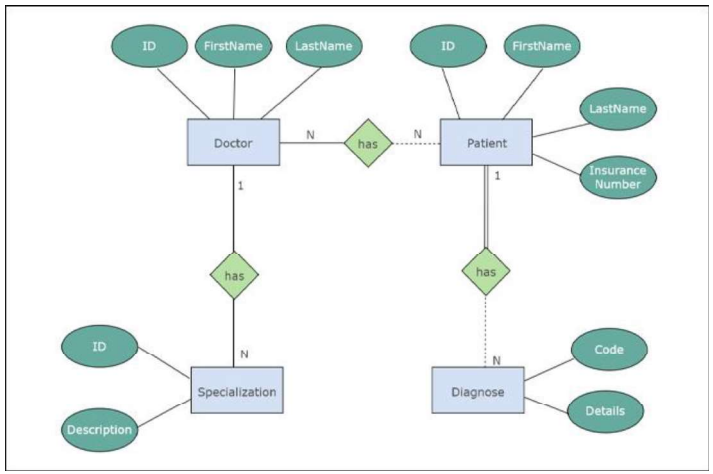
Example- Decision Tables

		Rules		
		Rule 1	Rule 2	Rule 3
Conditions	Calculation correct	No	Yes	No
	Decision template complete	Yes	No	No
	Telephone clarification sufficient	No	Yes	Yes
Actions	Telephone clarification	-	x	x
	Formulate and send query	x	-	-
	Correct calculation	x	-	x
	Correct decision template	-	x	x
	Sign decision template	x	x	x

Entity-Relationship Model

- Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model.
- ER Model creates a set of entities with their attributes, a set of constraints and relation among them.

Example- ERD



Data Dictionary

- Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc.
- Data dictionary is often referenced as meta-data (data about data) repository.

Elements of Data Dictionary

- Data Flow
- Data Structure
- Data Elements
- Data Stores
- Data Processing

Example- Data Dictionary

Data Dictionary

Data Dictionary outlining a Online Gaming Service

Data Item	Data Type	Data Format	Number of Bytes for Storage	Size for Display	Description	Example	Validation
MemberID	String	XNNNNNN	7	7	Unique Identifier For Member	M123456	
First Name	String		25	25	First Name of Member	Scott	
Surname	String		25	25	Last Name of Member	Daniels	
D.O.B	Floating Point (Date Format)	DD/MM/YYYY	4	10	Birth Date of Member	02/04/1990	Date < Today - 15 years
Platinum Membership?	Boolean	X	1	1	True (T) or False (F)	T	
Subscription Cost	Floating Point (Currency Format)	$NN.NN	4	6	Cost of Members Subscription	$27.50	Cost > 0 Cost < $50.00
Available Game Packs	Array (String)		25 * Number of Games	25 * Number of Games	Names of Game Packs	Open World Game Pack	

Discussion

HNDIT 4012 Software Engineering

Software Architecture

Topics covered

- Software Architectures
- Software Patterns
- MVC
- Layered
- Client Server
- Repository
- Pipe and Filter

What is Software Architecture?

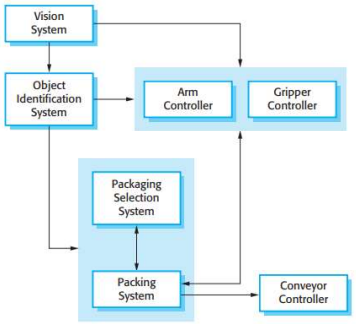
- Software architecture refers to the high-level structure or organization of a software system, which defines the components, relationships, and interactions that make up the system.

Key Aspects of Software Architecture

1. **Components:**
   - The software system is decomposed into modular components, each responsible for a specific set of functionalities or services. Components may include modules, classes, libraries, services, databases, and external dependencies.
2. **Connectors:**
   - Connectors define the communication and interaction mechanisms between components, facilitating data exchange, control flow, and coordination within the system. Examples of connectors include method calls, messages, APIs, and protocols.
3. **Architectural Styles and Patterns:**
   - Architectural styles and patterns provide reusable solutions to common design problems, guiding the selection and arrangement of components and connectors in the system. Examples include client-server, layered architecture, microservices, and event-driven architecture.

Example- Software Architecture

- The architecture of a packing robot control system.

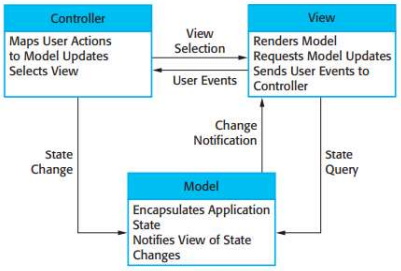


Architectural Patterns

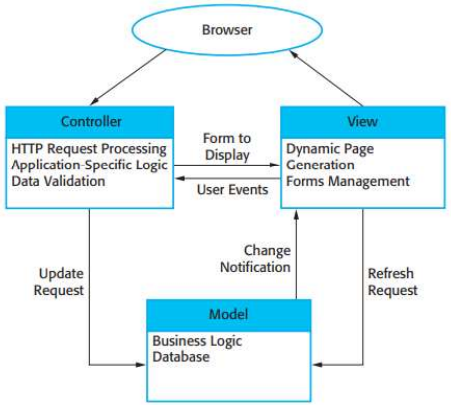
- You can think of an architectural pattern as a stylized, abstract description of good practice, which has been tried and tested in different systems and environments.

Model-View-Controller

- This pattern is the basis of interaction management in many web-based systems.



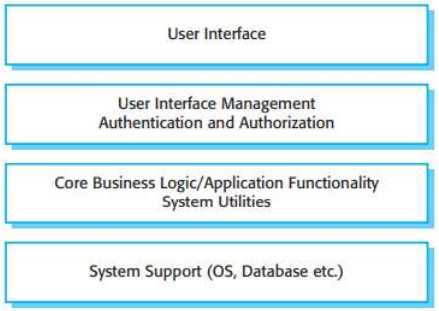
Example-MVC



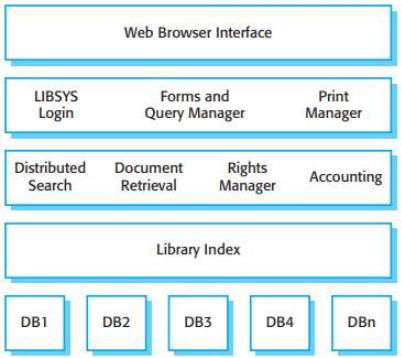
Layered architecture

- This layered approach supports the incremental development of systems.
- As a layer is developed, some of the services provided by that layer may be made available to users.
- The architecture is also changeable and portable.

Generic Layers



Example- Layered Architecture

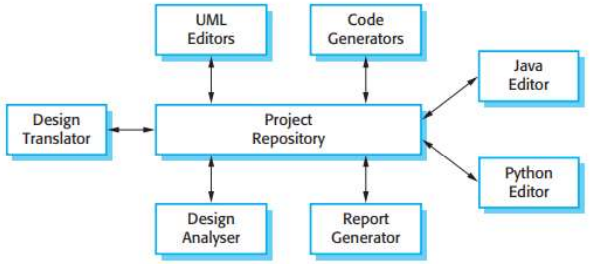


Repository Architecture

- All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.

Example- Repository Architecture

- A repository architecture for an IDE.

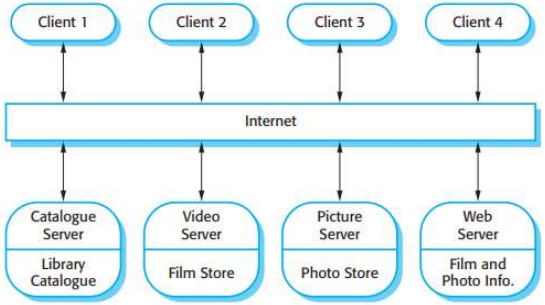


Client Server Architecture

- In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server.
- Clients are users of these services and access servers to make use of them.

Example- Client Server Model

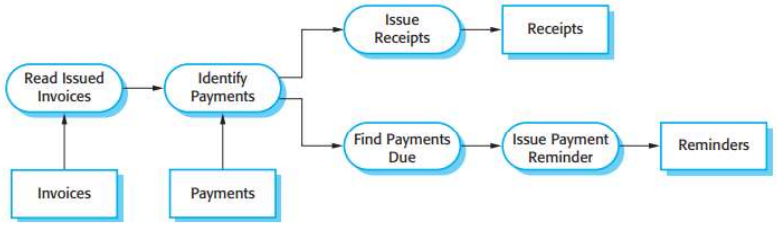
- This is a multi-user, web-based system for providing a film and photograph library.



Pipe and Filter Architecture

- This is a model of the run-time organization of a system where functional transformations process their inputs and produce outputs.
- Data flows from one to another and is transformed as it moves through the sequence.

Example- Pipe & Filter



Architectural Design Decisions

1. Is there a generic application architecture that can act as a template for the system that is being designed?
2. How will the system be distributed across a number of cores or processors?
3. What architectural patterns or styles might be used?
4. What will be the fundamental approach used to structure the system?
5. How will the structural components in the system be decomposed into subcomponents?
6. What strategy will be used to control the operation of the components in the system?
7. What architectural organization is best for delivering the non-functional requirements of the system?
8. How will the architectural design be evaluated?
9. How should the architecture of the system be documented?

Advantages

- Stakeholder communication
- System analysis
- Large-scale reuse

Discussion

- What is Service Oriented Architecture?
- SOA decomposes the application into loosely coupled, reusable services that expose functionality via standardized interfaces (e.g., web services).
- Services are designed to be independent, composable, and interoperable across different platforms and technologies.

HNDIT 4012
Software
Engineering

Software Implementation

Topics covered

- Coding
- Coding guidelines
- Code Review
- Code Inspection
- Code Walk through

What is Coding?

- The objective of the coding phase is to transform the design of a system into code in a high level language and then to unit test this code.
- The programmers adhere to standard and well defined style of coding which they call their coding standard.

Characteristics of a Programming Language

- Readability
- Portability
- Generality
- Error checking
- Cost
- Familiar notation
- Modularity

Coding standards and guidelines

- Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

Coding standards and guidelines

- 1. **Rules for limiting the use of global:** These rules list what types of data can be declared global and what cannot.
- 2 . **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Coding standards and guidelines

- 3.**Contents of the headers preceding codes for different modules:**
  - Name of the module.
  - Date on which the module was created.
  - Author's name.
  - Modification history.
  - Synopsis of the module.
  - Functions with their input/output parameters.
  - Global variables accessed/modified

Coding standards and guidelines

- 4.**Do not use a coding style that is too clever or too difficult to understand**: Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code.
- **5**. **The code should be well-documented**: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

Coding standards and guidelines

- **6. The length of any function should not exceed 10 source lines:** A function that is very lengthy is usually very difficult to understand as it probably carries out many different functions.
- 7. **Do not use an identifier for multiple purposes:** Each variable should be given a descriptive name indicating its purpose. This is not possible if an identifier is used for multiple purposes.

Comments

- This is one of the important coding components, as the comments included in the code describe what the code actually does and all other associated descriptions.
- This section also helps creating help documentations for other developers.

Code Review

- Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated.
- Code reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code.

Code Review

- Normally, two types of reviews are carried out on the code of a module.
  - Code inspection
  - Code walk through

Code Walk Throughs

- Code walk through is an informal code analysis technique.
- The main objectives of the walk through are to discover the algorithmic and logical errors in the code.
- A few members of the development team are given the code few days before the walk through meeting to read and understand code.

Code Inspection

- In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming.
- Good software development companies collect statistics regarding different types of errors commonly committed by their engineers and identify the type of errors most frequently committed.

Classical Programming Errors

- Use of uninitialized variables.
- Jumps into loops.
- Nonterminating loops.
- Incompatible assignments.
- Array indices out of bounds.
- Improper storage allocation and deallocation.
- Use of incorrect logical operators or incorrect precedence among operators.

Software Implementation Challenges

- Code-reuse
- Version Management
- Target-Host

Code Re-use

- Programming interfaces of present-day languages are very sophisticated and are equipped huge library functions.
- Still, to bring the cost down of end product, the organization management prefers to re-use the code, which was created earlier for some other software.

Version Management

- Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation.
- This documentation needs to be highly accurate and available on time.

Advantages

- The main advantages of adhering to a standard style of coding are as follows:
  - A coding standard gives uniform appearances to the code written by different engineers
  - It facilitates code of understanding.
  - Promotes good programming practices.

Discussion

- Software Implementation Challenges.
- Coding styles.

HNDIT 4012
Software
Engineering

Testing

Topics covered

- Introduction
- Verification & validation.
- Black-box Testing
- White-box Testing
- Manual & Automated Testing

Introduction

- The aim of the testing process is to identify all defects existing in a software product.
- Thus testing provides a practical way of reducing defects in a system and increasing the users' confidence in a developed system.

Aim of Testing

- **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

Software Validation

- Validation is process of examining whether or not the software satisfies the user requirements.
- It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

Software Verification

- Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

Testing Approaches

- Tests can be conducted based on two approaches .
  - Functionality testing
  - Structural testing

Functional Testing

- In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software.
- For this reason, black-box testing is known as functional testing.

Structural Testing

- On the other hand, in the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.

BLACK-BOX TESTING

- It is carried out to test functionality of the program. It is also called ‘Behavioral’ testing. The tester in this case, has a set of input values and respective desired results.

Black-box Testing Techniques

- Equivalence class portioning
- Boundary value analysis

Equivalence Class Partitioning

- In this approach, the domain of input values to a program is partitioned into a set of equivalence classes.
- Equivalence classes for a software can be designed by examining the input data and output data.

Example 01

- For a software that computes the square root of an input integer which can assume values in the range of 0 to 5000, there are three equivalence classes: The set of negative integers, the set of integers in the range of 0 and 5000, and the integers larger than 5000. Therefore, the test cases must include representatives for each of the three equivalence classes and a possible test set can be: {-5,500,6000}.

Boundary Value Analysis

- A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs.
- For example, programmers may improperly use < instead of <=.
- Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.

Example 02

- For a function that computes the square root of integer values in the range of 0 and 5000, the test cases must include the following values: {0, -1,5000,5001}.

White-box Testing

- It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as ‘Structural’ testing.

White-box Testing Techniques

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Path Coverage
- Control flow testing
- Data flow testing

Manual Testing

- This testing is performed without taking help of automated testing tools.
- The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.
- Manual testing is time and resource consuming.

Automated Testing

- This testing is a testing procedure done with aid of automated testing tools.

Discussion

- Software Testing Tools

HNDIT 4012
Software
Engineering

Testing Levels

Topics covered

- Testing Levels
- Unit Testing
- Integration Testing
- System Testing
- Testing Documentation
- Debugging
- Test Case

Testing Levels

- Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development.
- Software is tested on various levels –
  - Unit Testing
  - Integration Testing
  - System Testing

Unit Testing

- While coding, the programmer performs some tests on that unit of program to know if it is error free.
- Testing is performed under white-box testing approach.
- Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing

- The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module.
- The integration plan specifies the steps and the order in which modules are combined to realize the full system.

Integration Testing

- There are four types of integration testing approaches.
  - Big bang approach
  - Bottom- up approach
  - Top-down approach
  - Mixed-approach

System Testing

- System tests are designed to validate a fully developed system to assure that it meets its requirements.
  - Alpha Testing
  - Beta Testing
  - Acceptance Testing

Alpha Testing

- Alpha testing refers to the system testing carried out by the test team within the developing organization.

Beta Testing

- After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose.

Acceptance Testing

- This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

Performance Testing

- Performance testing is carried out to check whether the system needs the non-functional requirements identified in the SRS document.

Types of Performance Testing

- Stress testing
- Volume testing
- Configuration testing
- Compatibility testing
- Regression testing
- Recovery testing
- Maintenance testing
- Documentation testing
- Usability testing

Example-Stress Testing

- For example, suppose an operating system is supposed to support 15 multi programmed jobs, the system is stressed by attempting to run 15 or more jobs simultaneously.
- A real-time system might be tested to determine the effect of simultaneous arrival of several high-priority interrupts.

Testing Documentation

- Testing documents are prepared at different stages.
  - SRS document
  - Test Policy document
  - Test case document
  - Test report
  - Test logs
  - Test summary

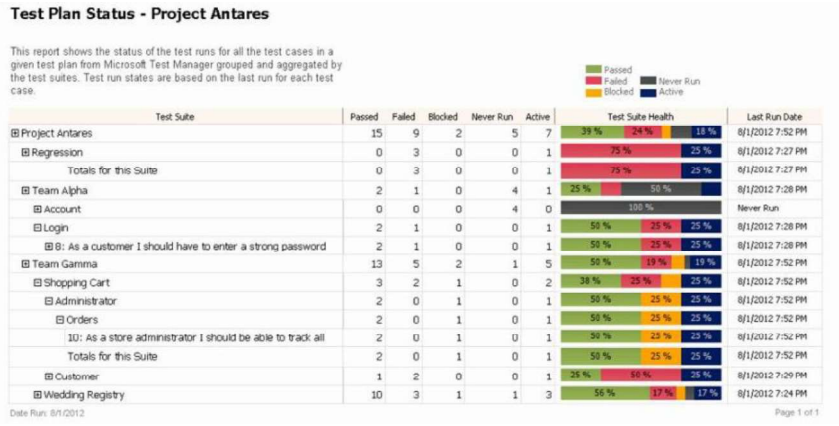
Sample Test Case

Test Scenario ID	Login-1	**Test Case ID**	Login-1A
Test Case Description	Login – Positive test case	**Test Priority**	High
Pre-Requisite	A valid user account	**Post-Requisite**	NA

Test Execution Steps:

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**	**Test Comments**
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter correct Email & Password and hit login button	Email id : test@xyz.com Password: ******	Login success	Login success	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Login successful

Sample Test Summary



Quality Assurance

- These are software development process monitoring means, by which it is assured that all the measures are taken as per the standards of organization.
- This monitoring is done to make sure that proper software development methods were followed.

Debugging

- Once errors are identified in a program code, it is necessary to first identify the precise program statements responsible for the errors and then to fix them.
- Identifying errors in a program code and then fix them up are known as debugging.

Debugging Guideline

- Debugging may sometimes even require full redesign of the system.
- One must be beware of the possibility that an error correction may introduce new errors.

Discussion

- Software Quality Control

HNDIT 4012
Software
Engineering

Software Quality

Topics covered

- Software Quality
- Quality factors
- ISO 9000
- CMM

What is Software Quality?

- Software Quality shows how good and reliable a product is. To convey an associate degree example, think about functionally correct software.
- It performs all functions as laid out in the SRS document.

Software Quality Factors

- Portability
- Usability
- Reusability
- Correctness
- Maintainability

Software Quality Activity

- The quality system activities encompass the following:
  - auditing of projects
  - review of the quality system
  - development of standards, procedures, and guidelines, etc.
  - production of reports for the top management summarizing the effectiveness of the
  - quality system in the organization.

Quality Assurance

ISO 9000 Certification

- ISO published its 9000 series of standards in 1987.
- In a nutshell, ISO 9000 specifies a set of guidelines for repeatable and high quality product development.
- The ISO 9000 standard specifies the guidelines for maintaining a quality system.

Types of ISO 9000

- ISO 9000 is a series of three standards: ISO 9001, ISO 9002, and ISO 9003.
- ISO 9001 applies to the organizations engaged in design, development, production, and servicing of goods.
- This is the standard that is applicable to most software development organizations.

Obtain ISO 9000

- Confidence of customers in an organization increases when organization qualifies for ISO certification.
- ISO 9000 requires a well-documented software production process to be in place.
- ISO 9000 makes the development process focused, efficient, and cost-effective.
- ISO 9000 sets the basic framework for the development of an optimal process and Total Quality Management (TQM).

SEI Capability Maturity Model

- SEI Capability Maturity Model (SEI CMM) helped organizations to improve the quality of the software they develop and therefore adoption of SEI CMM model has significant business benefits.
- SEI CMM classifies software development industries into the following five maturity levels.

CMM Levels

- Level 1: Initial - A software development organization at this level is characterized by ad hoc activities.
- Level 2: Repeatable - At this level, the basic project management practices such as tracking cost and schedule are established. Size and cost estimation techniques like function point analysis, COCOMO, etc. are used.

CMM Levels

- Level 3: Defined - At this level the processes for both management and development activities are defined and documented.
- Level 4: Managed - At this level, the focus is on software metrics. Product metrics & Process metrics.
- Level 5: Optimizing - At this stage, process and product metrics are collected. Process and product measurement data are analyzed for continuous process improvement.

Key process areas (KPA)

CMM Level	Focus	Key Process Ares
1. Initial	Competent people	
2. Repeatable	Project management	Software project planning Software configuration management
3. Defined	Definition of processes	Process definition Training program Peer reviews
4. Managed	Product and process quality	Quantitative process metrics Software quality management
5. Optimizing	Continuous process improvement	Defect prevention Process change management Technology change management

Parameters Defining Software Project

- **Cost:** As the main cost of producing software is the manpower employed.
- **Schedule:** This means that software needs to be developed faster and within the specified time.
- **Quality:** Developing high-quality software is another fundamental goal of software engineering.

Discussion

- Software Quality Control

HNDIT 4012
Software
Engineering

Software Maintainance

Topics covered

- Factors of Software maintenance
- Types of software maintenance
- Cost of maintenance
- Software re enginnering

Introduction

- Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product.

Necessity of Software Maintenance

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.

Necessity of Software Maintenance

- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company.

Types of maintenance

- Corrective Maintenance
- Adaptive Maintenance
- Perfective Maintenance
- Preventive Maintenance

Corrective Maintenance

- This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

Adaptive Maintenance

- This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

Perfective Maintenance

- This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

Preventive Maintenance

- This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

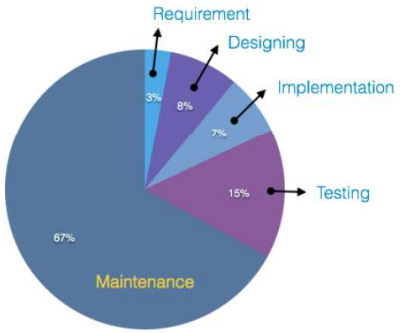
Problems associated with software maintenance

- Software maintenance work typically is much more expensive than what it should be and takes more time than required.
- In software organizations, maintenance work is mostly carried out using ad hoc techniques.
- Another problem associated with maintenance work is that the majority of software products needing maintenance are legacy products.

Estimate Cost of Maintenance

- Boehm [1981] proposed a formula for estimating maintenance costs as part of his COCOMO cost estimation model.
- Boehm's maintenance cost estimation is made in terms of a quantity called the Annual Change Traffic (ACT).

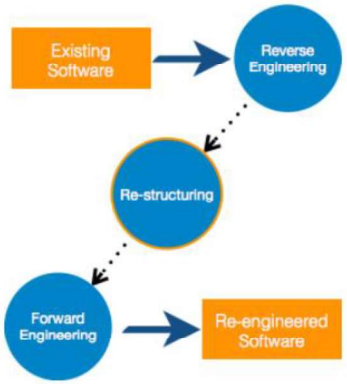
Cost of Maintenance



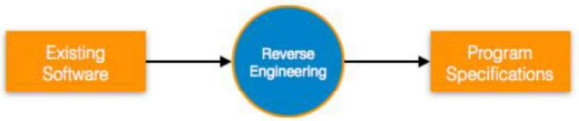
Software Re Engineering

- Software reverse engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code.

Re-Engineering Process



Reverse Engineering



Forward Engineering

- Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering.

Component Reusability

- In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.
- In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.
- There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).

Legacy Software

- It is prudent to define a legacy system as any software system that is hard to maintain.
- The typical problems associated with legacy systems are poor documentation, unstructured (spaghetti code with ugly control structure), and lack of personnel knowledgeable in the product.
- Many of the legacy systems were developed long time back

Discussion

- Configuration management