C1 : Introduction to Software Engineering

Software is a set of computer programs which are designed and developed to perform specific task desired by the user or computer itself.

Lecturer answer:
Software refers to a collection of instructions, programs and data that enable a computer system to perform specific tasks or functions.
It is a set of electronically encoded information that directs the operation of hardware components, such as processors, memory, storage devices and input/output devices.

Benefits of software

- Automation and efficiency
  - ↳ automates repetitive & time-consuming tasks
  - ↳ streamlines [illegible] processes
  - ↳ reduces manual effort
  - ↳ increases efficiency & productivity

- Accuracy and Reliability
  - ↳ eliminates human errors and inconsistencies
  - ↳ software executes tasks precisely and reliably if programmed correctly
  - ↳ improve data accuracy

- Scalability and Flexibility
  - ↳ can be scaled up or down to meet changing requirement
  - ↳ provides flexibility to adapt to evolving requirements (e.g. function expand, growth of user base, integrating with other systems)

- Improved Decision Making
  - ↳ software can collect, analyze, visualize data and provide insights for informed decision-making.
  - ↳ help organizations identify trends, patterns and opportunities

- Innovation and Competitive Advantage
  - ↳ helps organizations to develop new software applications, then introduce innovative products/services in market.
  - ↳ helps organizations to optimize operations
  - ↳ helps organizations to differentiate themselves in market.

- Improved Customer Experience
  - ↳ helps organizations to provide enhanced customer experiences
  - ↳ Customer relationship management (CRM) systems & customer service software enable better customer engagement, support & satisfaction.

- Cost Savings
  - ↳ automates tasks without requiring manual labour
  - ↳ reduces paperwork, optimizes resource allocation, eliminates need for physical storage
  - ↳ enables remote work, reduce office space & infrastructure expenses

- Accessibility and Conveniences
  - ↳ cloud-based software, mobile application & web-based platforms enable users to conveniently access across devices.

- Continuous Improvement and Updates
  - ↳ software can be updated regularly
  - ↳ improve functionality based on evolving needs
  - ↳ fix bugs
  - ↳ allows organizations to adapt and advance new technologies to deliver better user experiences.

Functions of Software

- Manage resources

  Manage computer resources of organization

- Provide tool

  Provide tools for human beings to take advantage of these resources

- Intermediary

  Act as an intermediary between organizations and stored information

Characteristics of Software

- Developed or Engineered

  Software is developed or engineered, it is not manufactured in the classical sense
  软件是开发或设计出来的，而不是传统意义上制造出来的

- Doesn't wear out (keep update / release new version)

- Continues to be custom built (customization)

System Software

- Collection of programs designed to operate, control and extend the processing capabilities of the computer itself.
- Prepared by computer manufacturers.
- function example : file editing, storage management, resource monitoring

Types of System Software

- System Control Programs
  - ↳ control execution of programs
  - ↳ manage storage and processing resources of computer
  - ↳ perform management & monitoring functions.

- System Support Programs
  - ↳ provide routine service functions to other computer programs & computer users
  - ↳ e.g. utility programs, antivirus

- System Development Programs
  - ↳ create publication programs
  - ↳ e.g. language translators like interpreters, compiler & assemblers

Application Software

- Computing software designed to carry out specific task other than relating to the operation of the computer itself, typically used by end-users.
- e.g. Microsoft Edge, Steam, Adobe Acrobat

Types of Application Software

- Generic products
  - ↳ stand-alone systems that are marketed and sold to customer who want to buy them
  - ↳ e.g. PC software such as graphics program, project management tools
  - ↳ 3 types:
    - a) Desktop App: applications that run in a personal computer and laptops.
    - b) Web App: applications that is accessed over a network such as the Internet or an intranet.
    - c) Mobile App: software application designed to run on smartphones, tablet computers and other mobile devices.

- Bespoke / Customized products
  - ↳ software that is commissioned by a specific customer to meet their own needs
  - ↳ e.g. embedded control systems, air traffic control software, traffic monitoring systems

Attributes of Good Software

- Availability : probability that a system will work as required during mission.

- Functional : system will perform the functions required.

- Efficient : system performs functions efficiently in terms of time and resources.

- Reliable : not easily cause hardware or software failure, deliver the functions required by users.

- Secure : protected against errors, attacks and loss of valuable data.

- Flexible : capable of being adapted to new uses, different countries or platform

- Buildable : Design is not too complex

- Manageable : Easy to estimate work & check progress

- Maintainable : Design helps maintenance programmer to understand the designer's intention

- Usable : Provide satisfying experience to users

- Reusable : Elements of system can be reused in other system.

Software Engineering

- An engineering discipline, which is concerned with all aspects of software production from early stages of system specification to maintaining system after it has gone into use.

- 4 layers :

  a) Quality Focus

  Any engineering approach must rest on organizational commitment to quality.

  b) Process

  defines a framework for the sequence and flow of software engineering activities to ensure effective delivery of software engineering technology.

  c) Methods

  organized ways of producing software, includes suggestion for the process to be followed, notations, rules governing the system description and design guidelines.

  d) Tools

  Provide automated and semi-automated support for the process and the methods.

Importance of Software Engineering

- Reliable product
- According to schedule
- Adapt to user needs
- Within budget
- Excellent performance
- Easy to maintain
- Cost-effective

Key Challenges of Software Engineering

维系

- Legacy challenge
  - ↳ maintain & update legacy software
  - ↳ avoid excessive cost
  - ↳ continue delivering essential business services.

- Heterogeneity / diversity challenge
  - ↳ build a dependable software which can adopt to different types of computer & support systems

- Delivery challenge
  - ↳ shortening delivery time for large & complex system without affecting system quality.

System Engineering

- An interdisciplinary 跨學科 field of engineering that focuses on how complex engineering projects should be designed and managed.

- Tutor answer:

  Software engineering is dealing with designing and developing the high quality software.

  System engineering is dealing with overall management of the engineering project during their life cycle.

C2 : Software Process Model

Process

- A series of steps taken to produce an intended output
- Steps involves :
  - ↳ Activities
  - ↳ Constraints
  - ↳ Resources
- Process involves :
  - ↳ Tools
  - ↳ Techniques

Characteristics of Process

- Prescribes 规定 all major process activities.
- Uses resources, subject to set of constraints (such as schedule, no. of people working)
- Produces intermediate and final products
- May be composed of sub-processes with hierarchy or links
- Each process activity has entry and exit criteria

Importance of processes

- Improve consistency and structure on a set of activities
- Guide us to understand, control, examine and improve the activities.
- Enable us to capture our experiences and pass them along 传递下去

Software Process Model

- A simplified description of a software process which is presented from a particular perspective.

- May include activities which are part of software process, software products and the roles of people involved in software engineering.

Software Process

- Software Specifications
  - ↳ Detailed description of a software system to be developed with functional and non-functional requirements.

- Software Development
  - ↳ designing, programming, documenting, testing & bug fixing

- Software Validation
  - ↳ evaluation software product
  - ↳ ensure the software meets business requirements / end users needs.

- Software Evolution
  - ↳ developing software initially, then timely updating it.

SDLC Models

- Process that produces software with the highest quality and lowest cost in the shortest time possible.

- Provides well-structured flow of phases

- Types :
  - ↳ Linear Sequential Model / Waterfall Model
  - ↳ Incremental model
  - ↳ Spiral model
  - ↳ Prototyping model
  - ↳ Rapid Application Model
  - ↳ Agile Software Development

Predictive VS Adaptive Software Methodologies

Predictive methodology

- ↳ requirements and schedule are known in advance
- ↳ project is planned and executed accordingly
- ↳ project was implemented in waterfall methodology
- ↳ known as plan-driven approach
- ↳ expensive modification
- ↳ high assurance 可信度
- ↳ designed for current & foreseeable requirements
- ↳ for large teams and products

Adaptive methodology

- ↳ requirements and schedule are not known in advance
- ↳ project is executed in an agile 敏捷 and iterative 递代 fashion
- ↳ known as Agile driven approach
- ↳ Inexpensive modification
- ↳ Rapidly deliver features
- ↳ designed for current requirements
- ↳ for smaller teams and products

Waterfall

- linear sequential flow
- progress is flowing steadily downwards through the phases of software implementation.
- any phase in development process begins only if the previous phase is complete.
- cannot go back to previous phase

Phases

① Requirements : define & plan project without mentioning specific processes

② Analysis : analyze system specifications to generate products models & business logic to guide production

③ Design : create design specification document to outline technical design requirements

④ Coding and implementation : develop source code using models, logic and requirement specifications designated in prior phases.

⑤ Testing : identify issues that must be resolved using quality assurance, unit, system and beta tests.

⑥ Operation and deployment : product/application is fully functional and deployed to live environment.

⑦ Maintenance : Carry out corrective, adaptive & perfective maintenance to improve product functionalities

When to use ?

- Fixed requirements
- Sufficient resources
- Established timeline
- Well-understood technology
- Not require significant changes

Advantages

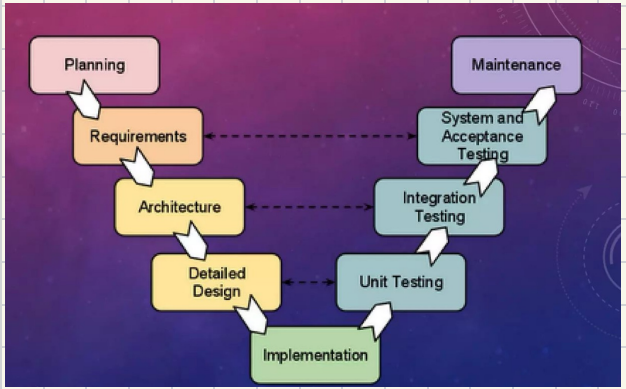
- enables large teams to move towards a common goal that's been defined in requirement stage
- forces structured, disciplined organization
- simplifies understanding, following and arranging tasks
- enable early system design and specification changes to be easily done
- clearly defines milestones 里程碑 & deadlines

Disadvantages

- design not adaptive (entire process has to be restarted once a flaw 缺陷 is found.)
- delay testing until the end of development lifecycle
- not ideal for complex, high risk projects
- no working products until the later stages of project lifecycle

V-Shaped

- extension for waterfall model
- process steps are bent upwards after coding phase, to form V shape.
- difference with waterfall : early test planning in V-shaped model.



When to use ?

- software requirements clearly defined and known
- software development technologies and tools is well-known

Advantages

- verification and validation of product in early stage of product development
- easy to use
- stages & activities are well defined
- high chance of success over waterfall (early test during lifecycle)
- each phase has specific deliverables
- work well (understandable requirements)

Disadvantages

- hard to handle if there are requirement changes during product development
- inflexible
- difficult to adjust scope (expensive)
- spend more cost & time
- require proper & detailed planning
- no continuous customer involvement

Incremental Model

- model requirements are broken down into multiple standalone modules of software development cycle.
- whole requirement is divided into various builds
- each module passes through:
    a) requirements
    b) design
    c) implementation
    d) testing

When to use?

- clearly defined & understandable requirements of complete system
- major requirements are defined
- introduce product into market early
- use new technology
- resources with needed skill set are not available
- there are some high risk features and goals

Advantages

- quickly generate working software
- flexible (less cost to change scope & requirements)
- easy to test & debug during small iteration
- customer can respond to each built
- low initial delivery cost
- easy to manage risk (identified & handled during iteration)

Disadvantages

- Needs good planning & design
- Needs clear & complete definition of whole system before it is broken down and built incrementally

Spiral Model

- known as cyclic model
- software is developed in a series of incremental releases
- more complete versions of system will be produced during later iterations
- Phases:
  a) Planning
  b) Design
  c) Construct
  d) Evaluation

When to use ?

- requires frequent deliverance
- large project
- unclear & complex requirements
- requires changes at any time
- large & high budget project

Advantages

- high amount of risk analysis
- useful for large and mission-critical projects

Disadvantages

- may spend more cost
- requires highly particular expertise for risk analysis 需要非常特殊的专业知识
- does not work well for small project

Prototyping Model

- before carrying out development of actual software, a working prototype of system should be built.
- toy implementation of system
- best scenario: only parts of project requirements are known in detail
- iterative, trial-and-error process between developers and users

Steps:

① Requirement Gathering and Analyst
② Quick Decision
③ Build a Prototype

(① ② ③ : Prototype development)

④ Assessment or User Evaluation
⑤ Prototype Refinement

(④ ⑤ : Iterative development)

⑥ Engineer Product

When to use?

- customers do not know the exact project requirements beforehand 事先
- developers are new to the domain
- repeatedly refine prototype based on customer feedback until a final acceptable prototype is achieved.

Advantages

- customers can see the partial product at early stages (customer satisfaction & comfortness)
- easily adapt to new requirements (scope of refinement)
- easily figure out missing functionalities (view partial product)
- errors can be detected early (save effort & cost)
- developed prototype can be reused for more complicated projects in future
- flexible design

Disadvantages

- too much changes in requirements each time the prototype is evaluated by customer
- poor documentation (continuous changing customer requirements)
- hard to accommodate 顾及 all changes demanded by customers
- can't determine the exact number of iterations required before the prototype is finally accepted by customer
- customers might demand actual product after seeing the early prototype
- customers might lose interest if he/she is not satisfied with the initial prototype

Types of model
- Rapid Throwaway Prototype
- Evolutionary Prototyping
- Incremental Prototyping
- Extreme Prototyping

Agile Methodology

- An approach to software development that seeks the continuous delivery of working software created in rapid iterations.

Principles of Agile Development

- customer satisfaction by rapid delivery of useful software.
- welcome changing requirements, even late in development
- working software is delivered frequently (in weeks)
- working software is the principal measure of progress
- sustainable 持续 development, able to maintain a constant pace
- close, daily co-operation between business people and developers
- face-to-face conversation is the best form of communication
- projects are built around motivated individuals, who should be trusted
- continuous attention to technical excellence and good design
- simplicity - the art of maximizing the amount of work not done
- self-organizing teams
- regular adaptation to changing circumstances

When to use ?

- constantly changing requirements
- teams have tight deadlines
- stakeholders want to reduce risk under deadlines
- teams can automate unit and functional tests

Advantages

- customer satisfaction (can check software release and revert feedback)
- meeting is arranged before product release
- more interaction maintained within developing and testing team
- customers can change / add requirements at any stage
- concentrates on every process with expert team members.

Disadvantages

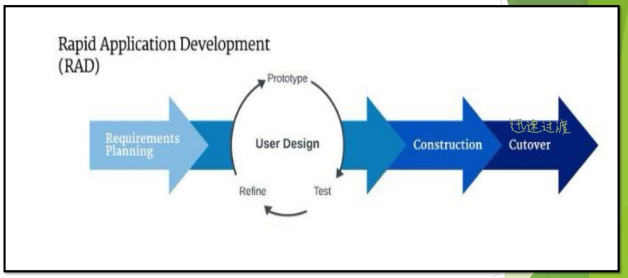
- lengthy documentation
- not useful for small projects
- requires expert persons to make important decisions in meeting
- more cost compared to waterfall / interactive model

Types :

- Extreme programming (XP) 🔗
- Feature Driven Development (FDD) 🔗
- SCRUM 🔗
- RAPID APPLICATION DEVELOPMENT (RAD)

RAD Model

- based on prototyping and iterative development with no specific planning involved.
- process of writing the software involves the planning required for developing the product.



When to use?

- system needs to create project that modularizes in a short span time (2-3 months)
- requirements are well-known
- limited technical risk
- needs to make a system which modularized in 2-3 months period
- budget allows the use of automatic code generating tools

Advantages

- Flexible for change
- Changes are adoptable
- Each phase brings highest priority functionalities to customer
- Reduce development time
- Increase reusability of features

Disadvantage

- Require highly skilled designers
- All application is not compatible with RAD
- Cannot use RAD model for small projects
- Not suitable for high technical risk projects
- Require user involvement

Process Model

** Filter out inappropriate model based on scenario given

- Waterfall
  - clear requirements
  - big project
- Incremental
  - Clear requirement (req analysis will not be iterated, only for design, implementation and testing)
  - Short stuff
  - Build by version (do core function)
- RAD
  - Clear requirements
  - Short dev time (3 months)
- Prototyping
  - Requirement not clear due client not computer literals.
- Spiral
  - High risk
- Agile
  - Rapid development
  - Short term by weeks

Incremental model

- Requirement always the same (planning, analysis)
- Technical skills can be changed in each increment. (design, implementation, testing)

- e.g.

  First iteration : food ordering using paper → staff records into system

  Second iteration : food ordering & records using tablet on each table

  Third iteration : food ordering & records via scanning QR code on table using customer's mobile phone

C3 : Project Management

- Project : problem scheduled for solution
- forces us to recognize that projects are aimed at solving problems and that failure to define the problem properly is what sometimes gets us into trouble.

- Project Management involves:
  (a) Planning
  (b) Monitoring
  (c) Control of People, Process and Events
- Project Management begins before any technical activity and continues throughout the Definition, Development and Support of Computer Software.

Importance to use Project Management in companies

- handle projects effectively
- define the project and agree with the customer
- plan and assess resource needs for project
- estimate project cost and make proposals
- plan & schedule activities in project
- allocate right resource at right time
- assess risk and failure points and make backup plans
- lead project team effectively and communicate well

Main Responsibility of IT Project Manager

- Planning

- Scheduling

  Planning & scheduling project development

- Supervise

  Supervise work to ensure that it is carried out based on required standards.

- Monitor progress

  To check that the development is on time and within budget.

Management Activities

- Proposal writing
  - ↳ describes objectives of project & how it will be carried out
  - ↳ includes cost & schedule estimations

- Project Planning
  - ↳ identifying activities, milestones & deliverables
  - ↳ project plan will be drawn to guide development towards the goals
  - ↳ e.g. Gantt Chart, PERT Chart

- Project Scheduling
  - ↳ separating total work involved into separate activities and judging the time required to complete these activities
  - ↳ set of charts showing:
    - (a) work breakdown
    - (b) activities dependencies
    - (c) staff allocations

- Project Costing
  - ↳ cost estimation: estimate resources required to accomplish project plan
  - ↳ estimate resources: hardware, software, people
  - ↳ cost-benefit analysis techniques: Payback period 回收期, Return on Investment (ROI) 投资回报率, Net Present Value (NPV) 净现值

- Project Monitoring
  - ↳ continuing project activity
  - ↳ keep track of project's progress
  - ↳ compare actual project progress with planned project progress
  - ↳ constraints:
    - (a) time
    - (b) cost
    - (c) quality
  - ↳ adopt corrective action based on situation
  - ↳ include informal monitoring & formal project reviews

- Personal Selection & Management
  - ↳ select skilled & experienced candidates to join project team
  - ↳ factors on staff selection:
    - (a) application domain & programming language experience
    - (b) attitude
    - (c) communication ability
    - (d) adaptability
  - ↳ Motivation to candidates:
    - (a) Money
    - (b) Job satisfaction
    - (c) Thoughtful job design
    - (d) Leadership

  - ↳ Factors that influence Group Working:
    - (a) Group Composition

      Is there the right balance skills, experience and personalities in the team?

      * Task-oriented / Self-oriented (do it yourself) / Interaction-oriented (communicate with other group members)

    - (b) Group Cohesiveness

      How to promote group cohesiveness?

      - ↳ establish group identity 群体认同
      - ↳ team building activities
      - ↳ members are treated as responsible & trustworthy

- Report Writing and Presentation
  - ↳ project managers report on the project
  - ↳ write concise 簡洁 and coherent 有条理 documents
  - ↳ effective communication :
    - (a) orally
    - (b) writing

(c) Group Communication

Do the members of the Group communicate effectively with each other ?

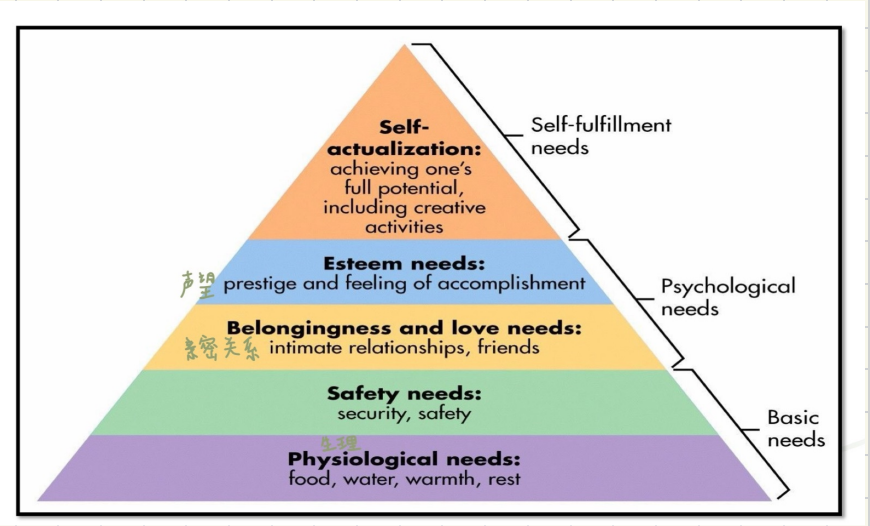
(d) Group Organization

Is the team organized in such a way that everyone feels valued & satisfied with their role in the group ?

↳ Assigned a right job to a right person

* Add on for Personal Selection & Management

Maslow's Motivation Model



5. Construct a ***Gantt Chart*** for developing an online system based on the following table.

Assume that the project starts in January.

Task ID	Task Description	Predecessor	Duration (month)	Overlap (month)
A	Form project teams and gather user requirements	None	¼	None
B	Plan project	A	½	None
C	Business modeling	B	1	¼
D	Data modeling	C	1	¼
E	Process modeling	C	1	¼
F	Application generation	E	1 ½	½
G	Testing & Turnover	F	½	¼

		January				February				March				April			
Task ID	Duration	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
A	¼																
B	½																
C	1																
D	1																
E	1																
F	1 ½																
G	½																

6. WildLife Wonderland is a veterinary, pet food and pet care shop. It is a small scaled family business founded in 1990s. Recently the shop owner, Anderson, plans to sell its products online. He has contacted your company, a software house, to handle this project.

Prepare a Gantt Chart to develop the online pet shop system based on the following table. Assume that the project starts in February.

Task ID	Task Description	Predecessor	Duration (month)	Overlap (month)
A	Requirement gathering	None	¼	None
B	Prototype 1	A	¼	None
C	Customer feedback	A	¼	None
D	Prototype 2	C	¼	None
E	Customer feedback	C	¼	None
F	Prototype 3	E	¼	None
G	Customer feedback	E	¼	None
H	Coding	G	1	None
I	Testing	H	½	½
J	Deployment	I	¼	None

		February				March				April			
Task ID	Duration	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
A	¼												
B	¼												
C	¼												
D	¼												
E	¼												
F	¼												
G	¼												
H	1												
I	½												
J	¼												

C4 : System Requirements

- Requirement specifies what the system should do & define constraints on its operation and implementation.

- 2 types of requirements :
  (a) User requirements
  (b) System requirements

User Requirements

- high level abstract requirement
- for user & people who procure 采购 system
- statements with natural language with diagrams & tables
- readers : client managers, contractor managers

* natural language : language where users can understand (e.g. English, Chinese, Malay)

User Requirement

1. The software must provide a means of representing and accessing external files created by other tools

System Requirements

- precise & detail descriptions of system functions
- written in structured form of natural language supported by system models and tables
- starting point of system design (aka functional specifications)
- basis of contract between system developer & customer
- readers : software developers

System Requirement

1.1 The user should be provided with facilities to define the type of external files

1.2 Each external file type may be represented as a specific icon on the user's display.

1.3 Facilities should be provided for the icon representing an external file type to be define by the user

Software System Requirements

- Functional Requirements
- Non-Functional Requirements

Functional Requirements

- statement of services the system should provide
- how system reacts to particular inputs
- how system behave in particular situation
- what system should not do

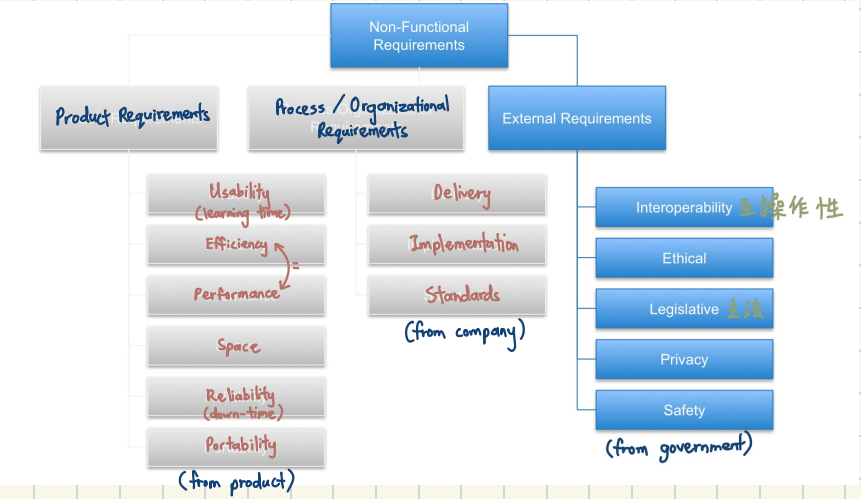
e.g. HR System - Employee Self Service Sub-system:

1.0 Basic Personal Tasks
- 1.1 Employees can perform routine HR administration tasks such as changing name, address, home / work telephone numbers, emergency contacts, next of kin, marriage, births or adoption details
- 1.2 Employees can upload common information to existing databases e.g. telephone directories and organization charts
- 1.3 Employees can view employees own employment history i.e. promotions, dates, appraisal reviews, training records
- 1.4 Employees can view conditions of employment, HR policies, procedures, information packs, benefits details

2.0 Time and Attendance Tasks
- 2.1 Employees can enter time sheets, plus with workflow, automatically submit request to manager for approval and once approved, automatically posted to payroll.
- 2.2 Employees can manage absences from work. Enter absence details.

Non Functional Requirements

- Apply constraints into requirements
- 3 types of non functional requirements:
  - ↳ Product Requirements
    - ↳ result from the need for the delivered product to behave in particular way
  - ↳ Organizational Requirements
    - ↳ consequences of organizational policies and procedures
  - ↳ External requirements
    - ↳ arise from factors external to system and its development process



Example of non functional requirements

- Example

Product requirement

*8.1 The **user interface** for LIBSYS shall be implemented as simple **HTML** without frames or Java applets.*

Organizational requirement

*9.3.2 The system development process and deliverable documents shall **conform** to the process and deliverables defined in XYZ-ST95.*

- Example

External requirement

*10.6 The system **shall not disclose any personal information** about system users apart from their name and library reference number to library staff who use the system.*

Problems with Natural Language

- Lack of clarity
  - ↳ hard to use language in precise & unambiguous 不含糊 way without making the document wordy & hard to read

- Requirements confusion
  - ↳ functional, non-functional requirements, system goals, design info may not be clearly distinguished
  - ↳ different requirement by different parties

- Requirements amalgamation 融合
  - ↳ several requirements expressed as a single requirement

Why natural language is not good?

- understanding of same word for same concept

- natural language is over flexible
  - ↳ sometimes one word can represent more than one meaning

- no clear partitioning
  - ↳ different people give different requirements as they are in different departments & have different expectations

System Requirement Specification (SRS)

Notation / Solution :

- Structured natural language
  - ↳ decision table
  - ↳ template / table for system input, process, output
- Design description language
  - ↳ like PL (e.g. PSL/PSA, RSL)
- Graphical notations
  - ↳ Data Flow Diagram (DFD)
  - ↳ Sequence Diagrams
- Mathematical specifications
  - ↳ Finite-state machine / set
  - ↳ Z- specification

Software Requirement Specification (SRS)

- A complete specification & description of the software that need to be fulfilled for the successful development of the software system.
- includes :
  - (a) official statement
  - (b) user requirements
  - (c) system requirements

Software Requirements Document

- specify what system should do without specifying how it should be done
- complete & consistent requirements
- specify all system functions
- requirements do not conflict

C5 : Requirements Engineering Process

Requirement engineering

- A process that involves all of the activities required to create and maintain a system requirements document.

- Process of identifying, eliciting, analyzing, specifying, validating and managing the needs and expectations of stakeholders for a software system.

5 main activities in Requirements Engineering Process

- Feasibility Studies
- Requirement Elicitation and Analysis
- Requirement Specification
- Requirement Validation
- Requirement Management

Feasibility Studies

- Technical feasibility

  inspects whether software can be built at all with available tools and experts.

- Economic feasibility

  inspects whether the proposed system will be cost-effective and if it can be developed using existing budget.

- Legal feasibility

  makes sure that the product complies with all regulations and doesn't break any law.

- Operational feasibility

  explores how a new project will impact daily processes in your company.
  what procedures and efforts should be taken to maintain it.

- Schedule feasibility

  analyze deadlines for proposed project which includes how much times teams will take to complete final projects

Requirement Analysis & Elicitation

- Process of deriving system requirements.

- Software engineers work with stakeholders to find out what application domain, services, hardware constraints, performance they want.

Difficulties

- Stakeholders not clear on what they want from the computer system

- Stakeholders express requirements in their own terms.

- Different stakeholders have different requirements and they may express them in different ways.

- Political factors may influence the system requirements.

- Economic and business environment may changes inevitably 无可避免 during analysis process.

Requirement Discovery

- Interact with stakeholders to discover requirements

- Source of info.
  - Stakeholders
  - Documentation
  - Similar system application

- Viewpoint-oriented Analysis
  - To get broad stakeholder coverage when discovering requirements
  - To classify stakeholders and other source of requirements

  ↳ Interactor Viewpoints
  - Interact directly with system
  - System features & interfaces

  ↳ Indirect Viewpoints
  - Do not use system
  - Influence requirements
  - High-level requirements & constraints

  ↳ Domain Viewpoints
  - Domain constraints

Add on : Pros & Cons of Viewpoint-oriented analysis

Pros :

- Multiple perspectives
  Captures the needs and requirements from different stakeholders or user groups, ensuring comprehensive coverage.

- Improve communications
  Facilitates better communication between developers and stakeholders by focusing on specific viewpoints.

- Conflict resolution
  Identify and resolve conflicting requirements early by addressing different perspectives

- Enhance traceability
  Allows better traceability of requirements to specific stakeholders, improving accountability and validation during development

- Structured approach
  Provides a systematic way to organize and categorize requirements, making the analysis more manageable.

Cons :

- Time-consuming
  spend more time on gathering and analyzing requirements from multiple viewpoints.

- Complexity
  Managing large number of viewpoints and conflicting requirements may add complexity to the process.

- Stakeholder engagement
  Requires active involvement from all stakeholders, which may not always be feasible, leading to incomplete or biased viewpoints.

- Integration difficulty
  Combining and harmonizing viewpoints into a cohesive system specification can be challenging.

Techniques of requirement discovery

- Interviewing

  open and closed questions may miss essential information

- Prototyping

- Scenario

  - find requirements by relating to real-life examples then abstract descriptions.
  - scenarios may written in text plus diagram.
  - general scenarios :
    - ↳ system & users expectation when scenario starts
    - ↳ normal flow of events
    - ↳ what can go wrong and how to handle
    - ↳ system state when the scenario ends

- Ethnography (observation)
  - an observational technique to understand social and organizational requirements.
  - discover implicit system requirements that reflect the actual rather than formal processes
  - focuses on end-users requirements only

Requirement Classification & Organization

- Organize requirements into clusters
- Remove overlapping requirements

Requirement Prioritization & Negotiation

- Prioritize requirements
- Resolve conflicts

* When multiple stakeholders are involved, the requirements will conflict. So, negotiation will be executed to prioritize requirements and finding / resolving conflicts.

Requirement Documentation

- Requirements will be documented and checked to discover if they are:
  - ↳ Complete
  - ↳ Consistent
  - ↳ In accordance with what stakeholders' requirements
- Formal and informal requirements document will be produced.

Requirements specification

Discussed in C4

Requirements Validation

- Process of checking that the defined requirements are for development, and defining the system that the customer really want.

Aspects to be checked:

- Validity : functions that user wanted is correct and confirmed
- Consistency : no conflicting functions
- Completeness : Include all functions needed
- Realism check : Could be implemented (with existing technology, budget, schedule)
- Verifiability : Are the requirements measurable? to reduce potential dispute 措在的争议

Requirement Management

- **Management planning**

  where policies and procedures for requirements management are designed and specified

- **Change management**

  where proposed system requirements changes are analyzed and their impacts are assessed.

C6 : Architectural Design

Architectural Design Activities

- System Organization
- Modular Decomposition
- Control Modeling

} * Have to refer lecnote for diagrams

Definition :

The initial design process of identifying sub-systems & establishing a framework for sub-system control and communication.

Advantages of Explicit Architecture

- Stakeholder communication

  Show high-level presentation of system during presentation

- System analysis

  Check whether the system can meet critical requirements

- Large-scale reuse

  Architecture may be reusable across range of systems since it shows us how a system is organized and how the components interoperate.

- Negotiation

  Serve as design plan for negotiation & discussion

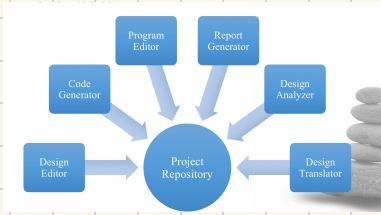
- Complexity Management

System Organization

- System is structured into many principal sub-systems
- Determine communication between sub-systems
- Show how sub-systems share data, being distributed, interface with each other

3 Widely Used System Organization Models:
- Repository Model
- Client Server Model
- Layered Model

Repository Model



When to use?
- Large amount of data to be shared

Sub-systems need to exchange data via:
(i) All shared data are stored in a central database.
(ii) Each sub-system maintain its own database and interchange data by passing messages

Advantages
- efficient way to share large amount of data
- Sub-systems need not be concerned with how data is produced
- Centralized management
- Sharing model is published as the repository schema.

Disadvantages
- Sub-systems must compromise 妥协 with repository data model inevitably 无可避免的
- Data evolution is difficult & expensive
- No scope for specific management policies
- Difficult to distribute efficiently

Client-Server Model

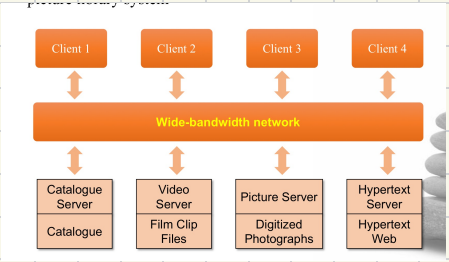
- Set of stand-alone servers which provide specific services
- Data and processing are distributed across range of components
- Set of clients call on services
- Network allows clients to access servers

Advantages

- Distribution of data is straightforward
- Effective use of networked systems. Cheaper hardware
- Easy to add servers or upgrade existing servers

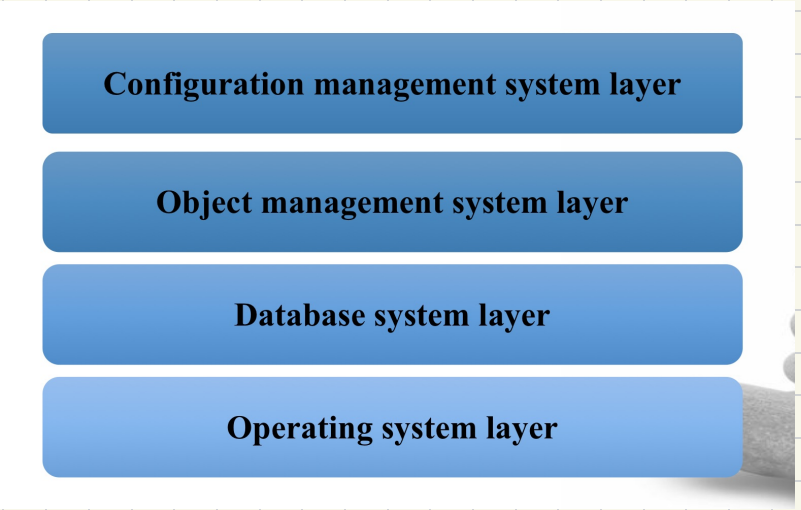
Disadvantages

- Different data organization, data interchange may be inefficient
- Redundant management in each server
- No central register of names and services



Layered Model

- To model the interfacing of sub-systems
- Organizes system into a set of layers, each of which provide a set of services
- Support incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer 相邻层 is affected.



Modular Decomposition

- Decomposition of sub-systems into modules.

2 Main Strategies :

- Object-oriented decomposition

  An object model where decompose a system into a set of communicating objects

- Function-oriented decomposition

  A pipeline / data-flow model where decompose a system into functional modules that accept input data and transform them into output data

Control Modeling

- Control sub-systems to ensure that their services are delivered to the right place at right time.

- Structural / Organizational models do not include control information.

- Establish a general model of control relationships between parts of system.

General control styles :

- Centralized control
  - ↳ Call-return model
  - ↳ Manager model

- Event-based model
  - ↳ Broadcast model
  - ↳ Interrupt-driven model

Centralized Control

- One sub-system has overall responsibility for control and starts and stops other sub-systems

Call-Return Model

- Top-down subroutine model
- Sequential systems

Manager Model

- One system manager
- Concurrent systems (多任务在同一时间执行)
- Sequential systems → case statement

Event-based Control

- Response to event from other system or environment

Broadcast Model

- Event is broadcast
- Whoever can handle will respond

Interrupt-Driven Model

- Interrupt Handler
- Real Time System

- Used in real-time system where interrupts are detected by interrupt handler and passed to some other component for processing.

C7 : Software Testing

Why testing ?

- Reliability

- Testing cannot show no errors in a program
- Testing should be part of the project planning

Main Goals of Testing

- Demonstrate to developer and customer that the software meets the requirements

- Discover faults and defects

Software testing stages

Different phases or levels at which testing is performed during System Development Lifecycle.

Stages in Testing Process :

- Unit Testing
- Module Testing
- Sub-system Testing
- System Testing
- Acceptance Testing (Alpha Testing)

Unit Testing

- Test individual components to ensure that they operate correctly
- Require test cases with condition and option

Module Testing

- Module is a collection of dependent components
- e.g. : Object class, Abstract data type, Collection of procedures / functions

Sub-system Testing

- Test collection of modules which have been integrated into sub-systems

* Combine a few modules, test whether it works or not

System Testing

- Test the integration of each sub-system in the entire system
- Find discrepancies 差异 between the system and its original objectives, current specifications and system documentation
- Primary concern is the compatibility of individual modules.

Acceptance Testing

- Final stage in testing process
- Aka user acceptance test (UAT) / alpha testing
- Conducted by users

Beta Testing

- Deliver a system to potential customers who agree to use that system
- Expose the product to real use
- Detect errors that may not be anticipated 预料 by developers

---

2 sources of test data:

- Live Data
  - ↳ actually extracted from organization files
  - ↳ difficult to obtain live data in sufficient amounts

  * get from company
  * e.g. real IC No, real staff name

- Artificial Data
  - ↳ Created solely for test purpose 仅为了测试目的而创建
  - ↳ Can be generated to test all combinations of format and values
  - ↳ Can be quickly prepared by data-generating utility program

  * dummy data

Testing Techniques and Strategies

2 fundamental testing activities:

- Component testing / Unit testing
  ↳ based on understanding of how components should operate & testing performed by software developers

- System testing
  ↳ integration testing
  ↳ release testing
  ↳ based on written system specification & testing performed by independent testing team

Software testing techniques

The specific methods, approaches or strategies used to design and execute test cases.

Integration Testing

Top Down Testing

- System testing
- Start from upper modules & works downwards

* Can be used when the lower modules are not ready

Bottom Up Testing

- System testing
- Start from fundamental modules & work upwards

Regression Testing

- System testing
- Rerun test for previous increments when a new increment is integrated
- If problems, check whether these problems are exposed:
  ↳ by new increment but actually ady existed in previous increment (not tested previously)
  ↳ due to added increment of functionality
- Expensive

Release Testing

Functional / Black Box Testing

- Relies on specification of the system or component, which is being tested to derive test cases.
- System behaviour determined by studying its inputs and outputs.

White Box Testing

- In release / integration testing phase
- Analyze code & use knowledge about the structure of a component to derive test data.
- To find many test cases to guarantee a given level of test coverage.

Performance / Stress Test

- Executed after system has been completely integrated
- To ensure system can process intended load
- Stressing system by going beyond its specified limits to test how well the system can cope with over-load situation

Component / Unit Testing

3 types of components to be tested:

- Individual functions / methods in an object
- Object classes that have attributes / methods
- Composite components that made up of different objects / functions

Individual Function / Method in an Object

- Simplest type of component
- Test : set of calls to these routines with different inputs parameters
- Used to design the test case

Object class

- The test should cover all features in an object
- The testing in isolation of all operations associated with the object
- The setting and interrogation 查询 of all attributes with the object
- Exercise of object in all possible states

Composite Component

- Primary concern : test component interface behaves according to its specification
- Interface errors cannot be detected in individual objects / components testing
- Errors may due to interactions between its parts.

Interface Testing

- to test composite components
- different types of interface error that can occur:
  - ↳ parameter interface
    data passed from one method to another
  - ↳ shared memory interface
    block of memory is shared between procedures / functions
  - ↳ procedural interface
    sub-system encapsulates a set of procedures to be called by other sub-systems
  - ↳ message passing interface
    sub-systems request services from other sub-systems

Thread testing / "Transaction flow" testing

- Used for systems with multiple processes where the processing of a transaction threads its way through these processes
- Event-based approach
- Identifying and executing each possible processing threads

Back-to-back testing

- Used when versions of a system are available.
- Different version of systems are tested together and their outputs are compared.
- Differences between test results can highlight potential problems

Testing Strategy

A general approach to the testing process, rather than a method of devising particular system or component tests.

- Proactive
  - ↳ think prevention steps early before problem came out
  - ↳ prevent bugs earlier before things getting worse
- Reactive
  - ↳ only find the bug / solution when the particular situation occurs.

Test Case Design and Planning

Test Plan Content

1. The testing process
2. Requirement Traceability
3. Tested items
4. Testing Schedule
5. Testing recording procedures (example)
6. Hardware & software requirements
7. Constraints

3 approaches for test case design:

- Requirement-based testing
  Test case designed to test system requirements are met

- Partition testing
  Identify input and output partitions and design tests

- Structural testing
  Use knowledge of the program structure to design tests that exercise all parts of the program

Testing Principles / Guidelines

- All tests should be traceable to customer requirements
  Most severe defects if the program fail to meet its requirements
  嚴重 缺陷

- Tests should be planned long before testing begins
  Test plan can begin once req. model is complete
  Test case can begin once design model has been confirmed

- The Pareto principles applies to software testing
  80% of all errors found during test will likely traceable to 20% of all program.
  So, isolate suspect components & thoroughly test them.

- Testing should begin "in small" and progress toward testing "in the large"
  First : testing focus on individual component
  Second : focus on finding errors in integrated components
  Third: focus on finding errors in the entire system

- Exhaustive testing is not possible
  impossible to test every combination of paths during testing because number of possible paths could be very large.

- To be most effective, testing conducted by independent third party
  developers may selectively test the parts that they think have defects.

C8 : User Interface Design

Human factors in interface design

- Limited short-term memory
- People make mistakes
- People are different
- People have different interaction preferences

Benefits of Good Design

- Small improvements can be worth big profits
- Interface problems are treated as bugs
- Big improvements can establish new products, companies, markets

Advantages

- Easy to learn and use
- User may switch attention between tasks and applications.
- Fast, full-screen interaction is possible with immediate access to the entire screen

Problems

- Many software systems are never used due to poor UI design
- A poorly designed UI can cause a user to make catastrophic errors

Errors in Using Colors

- Color is used to communicate meaning
  - ↳ Color-blind may misinterpret the color
  - ↳ Human color perceptions are different
- Too many colors or colors too bright
  - ↳ Not comfortable
  - ↳ Confusion

Guidelines of using color

- Don't use too many colors
- Use color coding to support user tasks
- Allow users to control color coding
- Design for monochrome then add color
- Use color coding consistently
- Avoid color pairings which clash
- Use color change to show status change
- Be aware that color displays are usually lower resolution

3 Golden Rules for a Good UI Design

- **Place the user in control**
  People do not like to be controlled by the machine and they like to know the status of the operation / system

- **Reduce the user's memory load**
  People have limited short-term memory & they make mistakes when they handle too much info.

- **Make the interface consistent**
  Easy to learn and the knowledge learnt in one command or application is applicable in other parts of the system.

UI Design Principle

- **User familiarity**
  should use the terms and concepts which are drawn from user experience

- **Consistency**
  wherever possible, operations should should be activated in the same way

- **Minimal surprise**
  users should never surprised by the behaviour of a system

- **Recoverability**
  should allow users to recover from errors

- **User guidance**
  provide help feature and meaningful feedback when errors occur

- **User diversity**
  should provide appropriate interaction facilities for different types of system user

C9 Real-time OS

Stimulus / response System

Type of stimuli :

- Periodic stimuli

  These occurs at predictable time intervals.

- Aperiodic stimuli

  These occurs irregularly (usually signaled using the computer's interrupt mechanism).

Stimuli and responses for a burglar alarm system

Stimulus	Response
Clear alarms	Switch off all active alarms; switch off all lights that have been switched on.
Console panic button positive	Initiate alarm; turn on lights around console; call police.
Power supply failure	Call service technician.
Sensor failure	Call service technician.
Single sensor positive	Initiate alarm; turn on lights around site of positive sensor.
Two or more sensors positive	Initiate alarm; turn on lights around sites of positive sensors; call police with location of suspected break-in.
Voltage drop of between 10% and 20%	Switch to battery backup; run power supply test.
Voltage drop of more than 20%	Switch to battery backup; initiate alarm; call police; run power supply test.

C10 : Software Evolution and Maintenance

Software Evolution

Software Evolution is the process of developing software initially, and then timely updating it for various reasons.

Software Maintenance

Software Maintenance is the process of modifying and updating a software system after it has been delivered to the customer

e.g. Software errors, installation of new hardware, customer needs

Types of Software Maintenance

- Corrective Maintenance
  A type of maintenance task / action performed after equipment failure.

- Adaptive Maintenance
  Modifying the software system to adapt it to changes in the environment, e.g. hardware platform, OS, business rules changes

- Perfective Maintenance
  Improve functionality, performance & reliability, and restructuring software system to improve changeability.

  e.g. New functional / non-functional requirements

Factors that Affecting Maintenance Cost

- Module independence
  high independency, low cost
- Programming language
  high level programming language
- Programming style
  Good structure → low cost
- Program validation & testing
  More time spent in testing, ↓ error, ↓ cost
- Documentation quality
  Good quality, good understanding, ↓ cost
- Configuration Management Techniques
  Effect technique → easy to keep track all versions, ↓ cost
- Application Domain
  New application domain, less understanding, ↑ cost
- Staff Stability
  New project reassignment, ↑ cost
  Same staff maintain, ↓ cost
- Age of the program
  > older, > maintenance it receives, > cost
- Dependence of program on its external environment
  > depend on environment changes, > cost
- Hardware stability
  If hardware no need to change, < cost (rarely happen)

Software Reengineering

Objectives:

- Improve system structure
- New system documentation
- Easy understand system
- Re-implement legacy system to make them easy to maintain

How?

- Source code translation
- Reverse engineering
- Program restructuring
- Program modularization
- Data reengineering

Source code translation

Translate source code into another programming language

↳ hardware platform update

↳ staff skill shortage

↳ organizational policy change

↳ CASE tool, e.g. REFINE

↳ still need manual work

Reverse Engineering

Derive system design & specification → documentation

↳ documentation → input to Program Modularization

↳ documentation → input to requirement specification for program replacement

↳ documentation → help program maintenance

Program structure improvement

Control structure

↳ simplify complex conditions

Program Modularization

Group related parts

↳ remove redundancy

↳ architectural transformation

Data Re-engineering

Problems with data before re-engineering:
- data naming problems (file / attributes)
- field length problems
- record organization problems
- hard coded literals / values
- no data dictionary

Problems with data value inconsistencies before re-engineering:
- inconsistent representation semantic 語意
- inconsistent handling of negative values

Cost Factors of Re-engineering:
- extent of data conversion
- expert staff
- tool support
- software quality

Advantages of Re-engineering
- reduce effort
- reduce development time
- reduce cost
- reduce risk

Disadvantages of Re-engineering
- tools support
- knowledge engineers
- practical limits

Legacy System Management

Legacy system assessment