System Dev Exam Notes

Systems Development (Monash University)

Week 1 – Introduction to Systems Development

Software Development

Computer Application

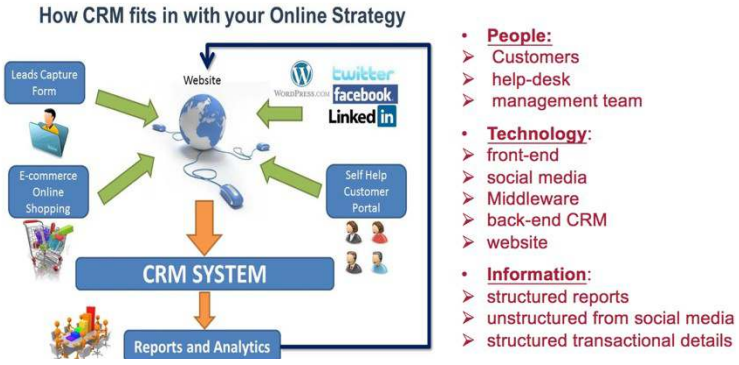
- Computer software program that executes on a computing device to carry out specific set of functions
  - Modest scope

Information System

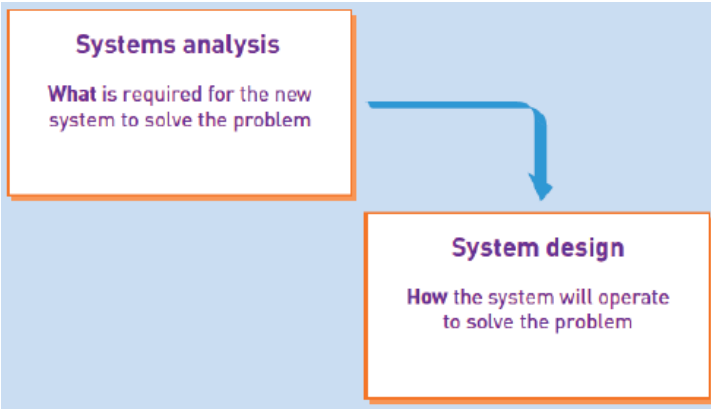
- Set of interrelated components that collects, processes, stores and provides as output the information needed to complete business tasks
  - Broader in scope than "app"
  - Includes databases/manual processes
  - Needed to: manage operations, interact with customer/supplier and compete in marketplace

What are Information Systems?

- Integrated set of components for collecting, storing and processing data for delivery of information
- Main component of an info sys (which interact with each other)
  - People
  - Procedures
  - Hardware and software
  - Databases
  - Data warehouses
  - Telecommunications



Systems Development: collection of analysis, design, coding, testing and documenting



System Analysis

- Those activities that enable a person to *understand and specify* what an information system should accomplish (identify stakeholders, tasks they want performed, document the tasks and the priority)

System Design

- Those activities that enable a person to *define and describe in detail* the system that solves the needs

System Development Life Cycle (SDLC)

- The process consisting of all activities required to; build, launch and maintain an info sys
- Six core process are:
  1. Identify problem or need and obtain approval
  2. Plan and monitor the project
  3. Discover and understand details of problem or need
  4. Design the system components
  5. Build, test and integrate system components
  6. Complete system tests and deploy solution

Project

- Planned undertaking that has a beginning and end and that produces some definite result
  - Used to develop an info sys
  - Requires knowledge of system analysis and design tools and techniques

System Development Process

- Actual approach used to develop a particular info sys
  - Unified process (UP)
  - Extreme Programming (XP)
  - Scrum

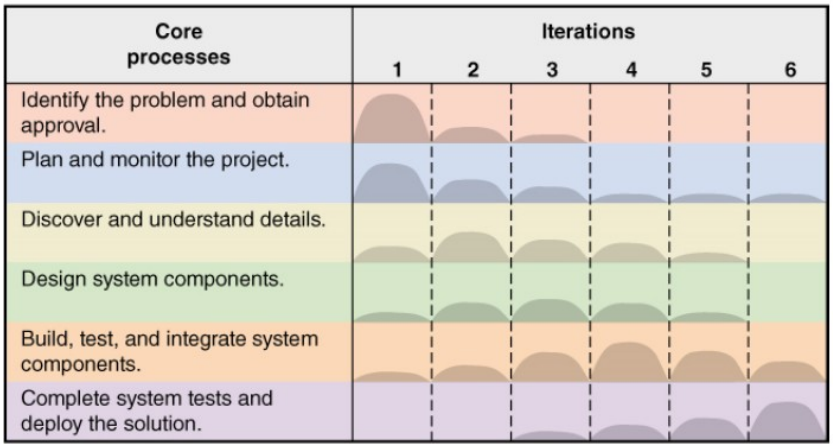
Iterative Development

- Approach to Sys Dev in which the system **grows in piece by piece through multiple iterations**
  - Complete small parts of system, then repeat processes to refine and add more, then repeat to refine and add more, until done

Agile Development

- An info sys dev process that **emphasizes flexibility to anticipate new requirements during development**
  - Fast on feet; responsive to change

Iterative and Agile Systems Development Lifecycle(SDLC)



Processes

1. **Initiation: Feasibility - Can it be done?**

- Can you afford to build what you want?
- Time constraints?
- Expertise available
- Willing to compromise
- Mandatory vs option
- Time + Budget

Initiation - Planning your development project

- Scope will stay fixed?
- How to manage scope creep?
- Project planning

2. **Analysis: What do you want?**

- Does the client know what they want?
  - Determines how you go about the process
  - It is vital that you demonstrate to client that you understand their requirements
- E.g.
  - Client requirement: Modern 4-bedroom house with 2 toilets and a garage
    - Which house does the client want?
    - The house together with a 1000 other houses would meet the brief?

Analysis - Build or Buy?

- Do you have to build or can you just buy?

3. **Design: How are you going to do it?**

- Detailed plans for the build
  - Shows integration of various components
    - Plans for carpenters, electricians, plumbers

Some tasks include:

- T1= identify major hardware/software subsystems/components

T2= decide on hardware/software platform for new system
T3= design IS software, database/UI
T4= security considerations

4. **Implement: Build/Develop - Construct, Test**

- Good analysis and design is essential for good build
- Good together with building expertise and thorough testing
- Not however just through building expertise only

Implement - Deploy: Is it Ready? Can I move in now?

- Does it meet
  - Government requirements
  - Sustainability requirements
  - CLIENT requirements
- Are your clients happy? Very costly exercise if the requirements are not met

5. **Support: Maintain it, Extend it**

- Can it be easily maintained and fixed?
- Can it be added to easily?

System Development Roles

Managerial - PM + TL
Functional - System + Business Analyst, Tester, UX Designer
Technical - System Designer, Database Admin,
Other - Quality Assurance, Documentation, Training and Deployment

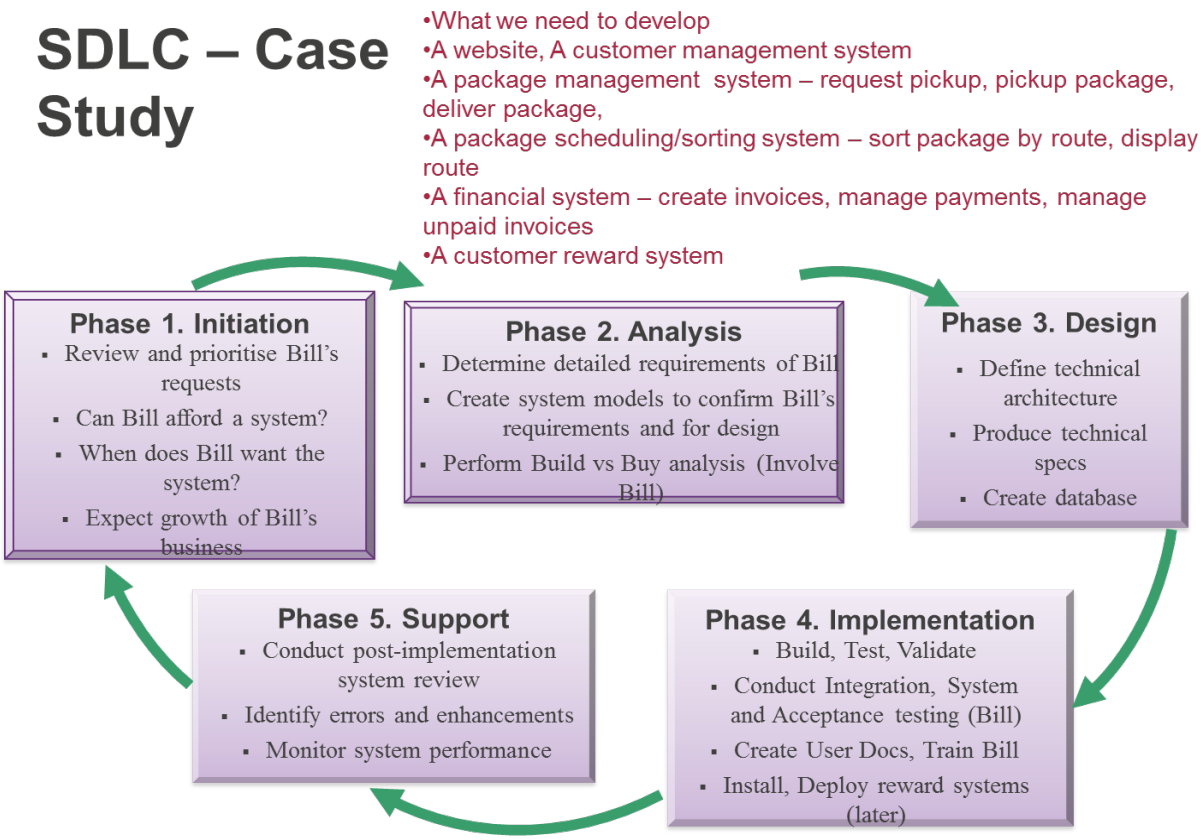
System Developer Role

- **Understanding Business**
  - Awareness and sensitivity to the business processes and needs that require technology in the first place
- **Broad and up - to - date understanding of technology**
  - Can be invaluable in creating the "best" solution for the organisation
- **Multiple Perspective**

- Ability to understand that there are multiple perspectives to solving problems is required to find the best solution

- **People/Soft Skills**
  - Ability to interact with other people and to be a part of a team
- **Continuous Learning**
  - Essential to a high change industry, like IT

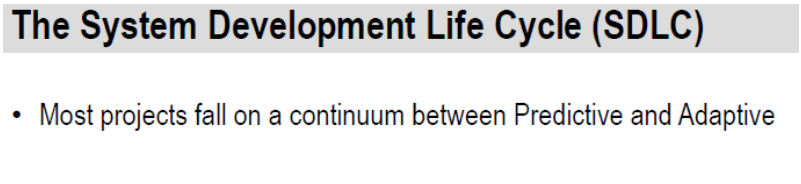
- Bill Wiley – start up, same day courier service
- "Initially just received delivery requests via texts on his mobile, but then customers started asking if he had a website where they could place orders. As the business grew, Bill hired another person to help with the deliveries. He could no longer use his van as the 'warehouse', he now needed a central warehouse where he could organise and distribute packages for delivery, and if it grew further someone at the warehouse to co-ordinate the arrival and distribution of the packages."
- What do you need to do to develop a system for Bill Wiley?

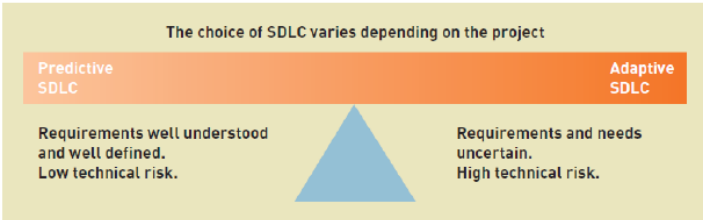


Week 2 – Approaches To Systems Development

SDLC

- Two general approaches:
- Predictive Approach (Stable/predictive & Low risk)
  a. Waterfall model
  b. Assume project can be planned in advance + info system can be developed according to the plan
  c. Requirements are well understood, stable and/or low technical risk

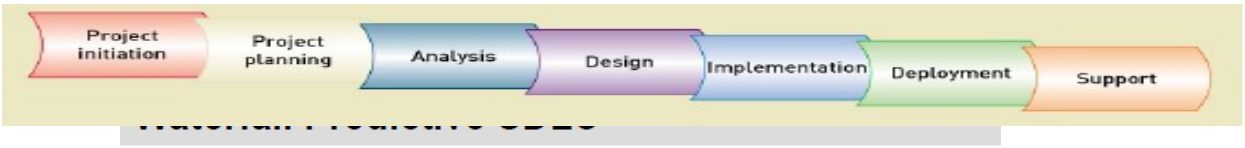


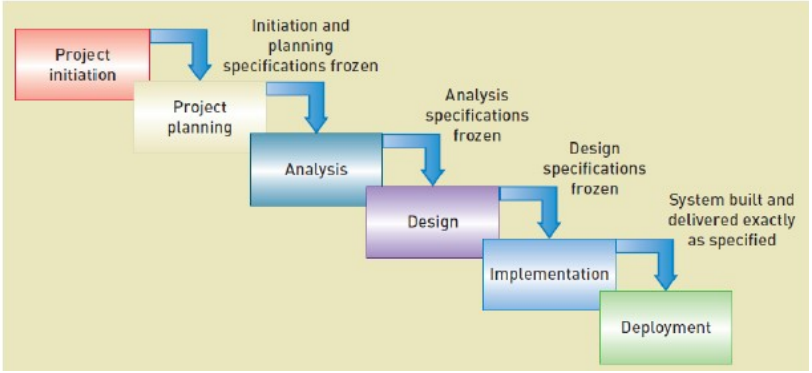


- Adaptive approach (Uncertain/Unstable & High risk)
  a. Iterative model
  b. Assume project is flexible and adapt to changing needs as project progresses
  c. Requirements and needs are uncertain and/or high technical risk

Traditional Predictive SDLC

- Earlier approach based on engineering
- Typically have sequential *phases*
  a. Phases are related groups of development activities, such as planning, analysis, design, implementation, and deployment
- Waterfall Model
  a. SDLC that assumes phases can be completed sequentially with no overlap/iteration
  b. Once phase is completed, you fall over the waterfall to next phase, no going back



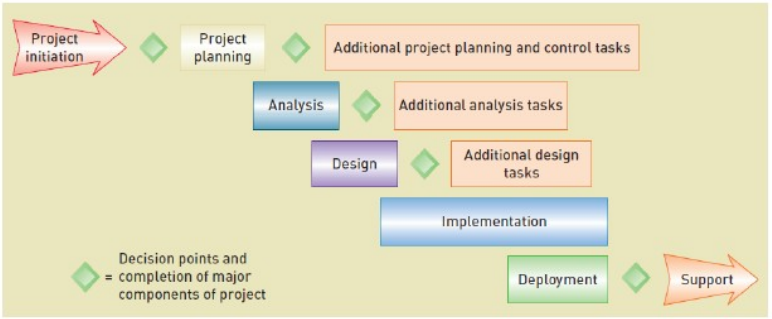


Newer Adaptive SDLC

- Emerged in response to increasingly complex requirements and uncertain technological environment
- Always included iterations where some design and implementation is done from the beginning
- Many developers claim it is the only way to develop info sys
- Flexibility is added due to recognising need for overlaps, new needs arise to revisit requirements in details (but still assumes predictive planning/sequential phases) but IT managers sometimes sceptical due to lack of overall plan

Modified Waterfall with Overlapping Phases

- More flexibility, but still assumes predictive planning and sequential phases



Incremental Development

a. Completes portions of the system in small increments (done in iterations) and integrated as the project progress
b. Sometimes considered growing a system

- Walking skeleton

a. The complete system structure is built first, but with bare-bones functionality

Incremental development	Walking Skeleton
Always based on an iterative development life cycle	Always based on iterative development life cycle
System is built in increments	A complete system but with only bare functionalities
An increment may consume 1 or more iterations	Early iterations develop the walking skeleton
An increment is integrated with the whole system	Later iterations add more details
Users get to see the system earlier	Users also get the system early

Iterative or Evolutionary Development

Fundamentally different philosophy to SDLC

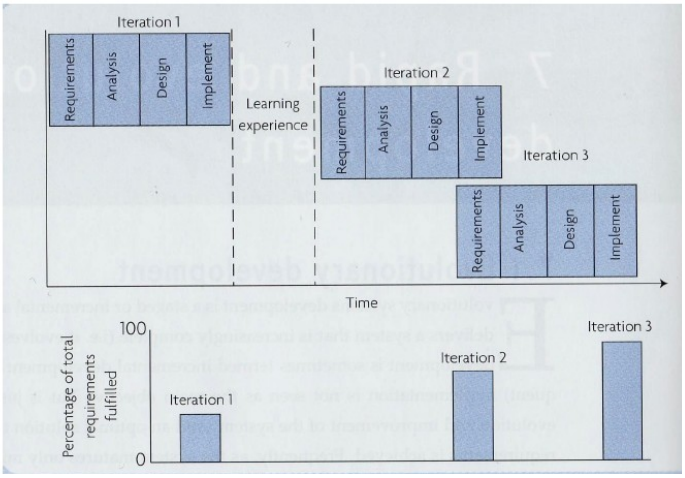
Iterative development

Staged or incremental approach

a. Design is not perfect
b. Accommodates change
c. Requirements not frozen
d. Does not have to be comprehensive

- Normally a period of learning between each stage
- Most useful when requirements are unclear

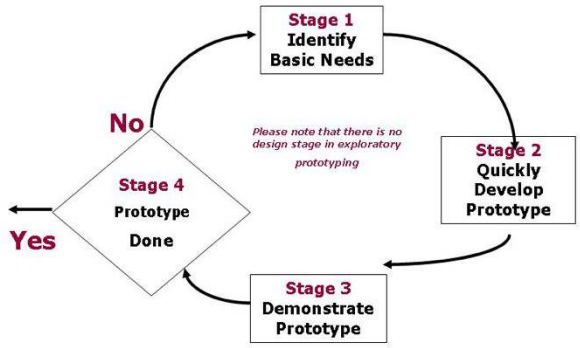
Evolutionary Development



Prototyping – "First Model"

- "Users only see their Info sys at implementation time when it is too late to make changes" Models are first produced (various experiments conducted by designers) before actually manufacturing products for commercial purposes – in software app development however:
- Two main types of prototype
  - **a. Interface prototyping (Facades)**
    - User interactions/preferences for interface are UNCERTAIN, but requirements STABLE
    E.g. range of end users like blind, deaf, access controls
  - **b. Exploratory software prototyping (Working)**
    - User requirements identified by developing series of working prototypes with simulated data
    Assumes: users don't have clear understanding of future systems, and once users interact with prototype it becomes clear
    Speedy development needed: refresh memory of end users, help them recall comments of previous demonstration of prototype
    Enthusiasm of users wanes if time lag

    Conceptual model:



- Issues of exploratory prototyping:

1. If fully functional working prototype meets user requirements, temptation to install
2. Tendency to discredit developers from first prototype and broadcast incompetence to capture needs
3. Difficulty in deciding when to stop iterations in prototyping (cost and time)
    - Inadequate and incomplete system designs
    a. Quick and dirty
    - Difficult to manage and control
    a. Poor documentation
    - Can become the system without proper engineering

Prototyping Methods

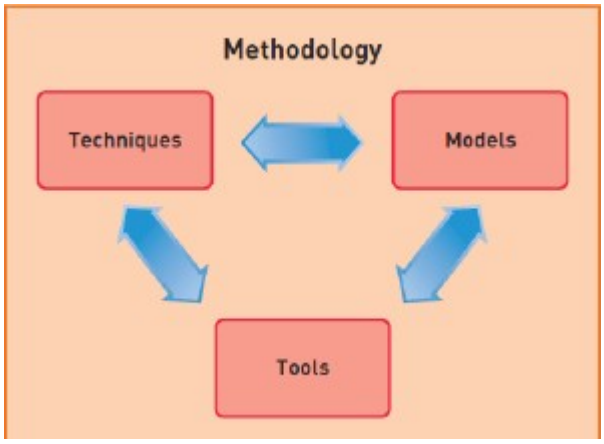
- Analysis Phase
  a. Understand current system and suggest improvements and functional requirements
- Prototyping Phase
  a. Construct a prototyping for evaluation by user
- Evaluation and prototype modification stages
  a. Could be multiple stages
- Design and develop the Target System
  a. Prototype is basis of specification

A Generic Adaptive Approach

- Six core processes go across iterations
- Multiple iteration required

Methodologies, Models, Tools, and Techniques

- Methodologies
  a. Provides guidelines for every facet of sys development
  b. Specifies an SDLC with activities and tasks
  c. Specifies project planning and project management models and reporting
  d. Specifics analysis and design models to create
  e. Specifies implementation and testing techniques
  f. Specifies development and support techniques



Methodology

- Includes a collection of models, techniques/tools that are used to complete activities and tasks, including modelling, for every aspect of the project
-

- Model
  a. An abstraction of an important aspect of the real world
  b. Each model shows a different aspect of the concept
  c. Crucial to communicate with end users

In IS, models are of system components that will be developed
Other models are used to manage the development process

Some models of system components
- Use case diagram
- Domain model class diagram
- Design class diagram
- Sequence diagram
- Package diagram
- Screen design template
- Dialog design storyboard
- Entity-relationship diagram (ERD)
- Database schema

Some models used to manage the development process
- Gantt chart
- Organizational hierarchy chart
- Financial analysis models—NPV, payback period
- System development life-cycle model
- Stakeholders list
- Iteration plan

- Tools
  a. Software app that assists developers in creating models or other components required for a project

Integrated Development Environment
  b. Set of tools that work together to provide a comprehensive development environment and produce source code/reverse engineering tools

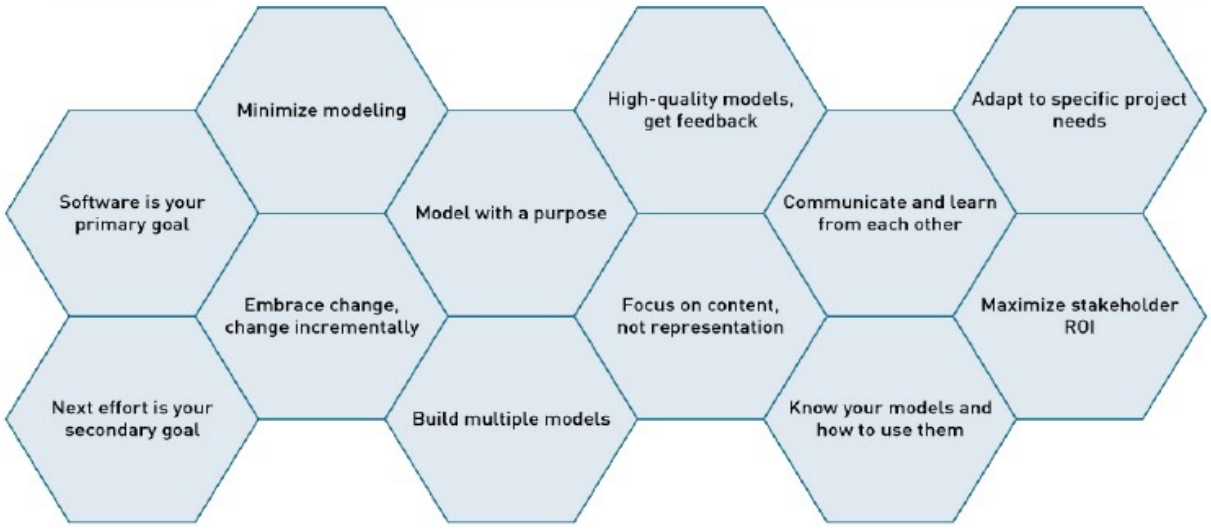
Visual modelling tools
  c. Tools to create graphical models

- Techniques
  a. Collection of guidelines that help an analyst complete an activity or task
  b. Learning techniques is the key to having expertise in a field

Agile Development

- Guiding philosophy and set of guidelines for developing info system in an **unknown, rapidly changing environment**
- **Willingness/ability of software development team to accept and respond to changes.**
- Chaordic
  a. Term for adaptive project - chaotic yet ordered
- Agile Values
  a. Values responding to change over following a plan
  b. Values individuals and interactions over processes and tools
  c. Values working software over comprehensive documentation
  d. Values customer collaboration over contract negotiation

- Agile Modelling (AM) – 12 Principles
  - A philosophy – build only necessary models that are useful and at the right level of detail



Agile Principles

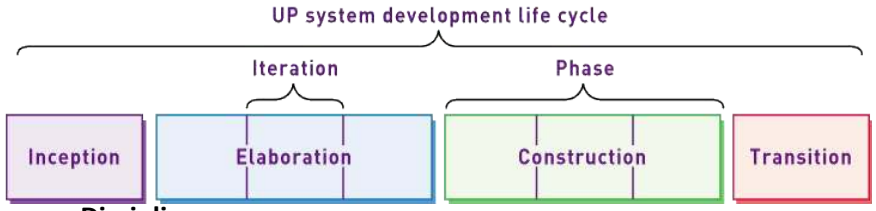
- **Principle 1:** Develop software as primary goal
  a) don't get distracted by documentation or models
- **Principle 2:** Enable the next effort as your secondary goal
  a. Be aware of next step versions or revision/support and maintenance
- **Principle 3:** Minimize your modelling activity
  a. Only build what helps move the project forward
- **Principle 4:** Embrace change and change incrementally
- **Principle 5:** Model with a purpose
  a. Model to understand
  b. Model to communicate
- **Principle 6:** Build multiple models
  a. Look at problem from different perspective
- **Principle 7:** Build high-quality model and get feedback
- **Principle 8:** Focus on content rather than representation
  a. Informal hand-drawn models are sometimes ok
  b. Always focus on stakeholder needs
- **Principle 9:** Learn from each other with open communication
- **Principle 10:** Know your model and how to use them
- **Principle 11:** Adapt to specific project needs
- **Principle 12:** Maximise stakeholder ROI

UP Phase	Objective
Inception	Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimates for cost and schedule.
Elaboration	Define the vision, identify and describe all requirements, finalize the scope, design and implement the core architecture and functions, resolve high risks, and produce realistic estimates for cost and schedule.
Construction	Iteratively implement the remaining lower-risk, predictable, and easier elements and prepare for deployment.
Transition	Complete the beta test and deployment so users have a working system and are ready to benefit as expected.

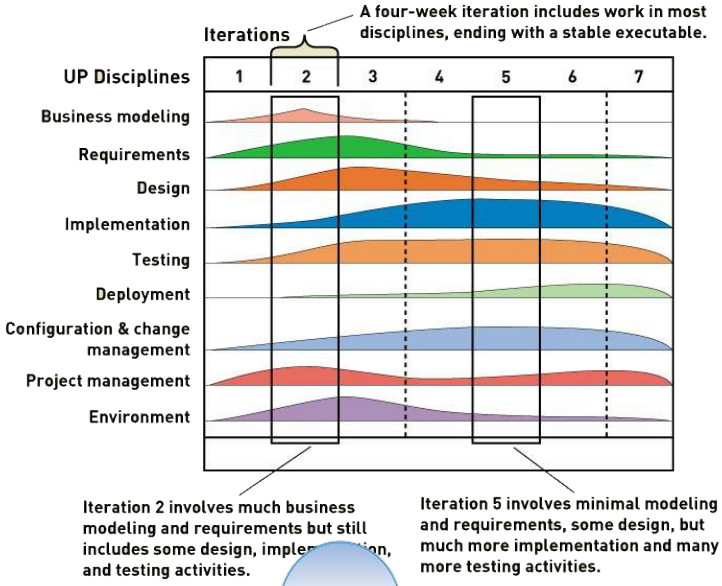
E.g. of Agile methodologies:

Unified Process (UP)

- UP & UML (Unified Modelling Language) were developed together – object oriented
- Phases (UP Lifecycle)
  a. UP Phases organize iterations into 4 primary areas of focus during a project – iterations differ in each phase
    - Inception - getting project started
    - Elaboration - understanding system requirement
    - Construction - building system
    - Transition - prep for and moving to deploy new system



- **Disciplines**
  a. Set of functionally related development activities
  b. Each discipline are all the activities related to achieving one objective in the development project
  c. Two types of disciplines
    - Systems development activities
    - Project management activities
      - Planning and control disciplines
  d. Software Development D
    - Business modelling
    - Requirements
    - Design
    - Implementation
    - Testing
    - Deployment
  e. Project management
    - Configuration and change management
    - PM
    - Environment

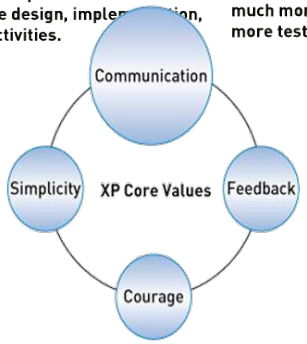


Extreme Programming (XP)

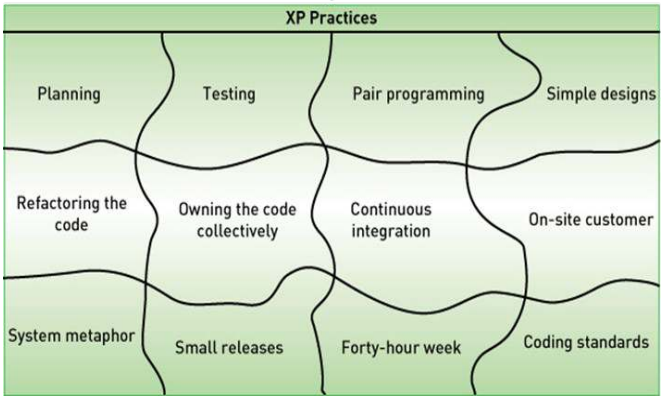
Takes the best practice of software dev and extends them "to the extreme"

  f. Focuses intensely on proven industry practices
  g. Combines them in unique ways to get better results

- **XP core values**
  a. Communication
  b. Simplicity
  c. Feedback
  d. Courage



- **XP Practices**
  a. 1. Planning - based on user stories
  b. 2. Testing - thorough testing at each step
  c. 3. Pair Programming – divides coding (algorithm design, writing code, switch)
  d. 4. Simple Designs- Agile modelling principles
  e. 5. Refactoring - redo and clean-up as you go
  f. 6. Owning the code collectively - egoless dev, anyone can review and improve code
  g. 7. Continuous integration - grow the software continuously
  h. 8. Onsite customer - get sign-off as you go
  i. 9. System Metaphor - what should the final system look like

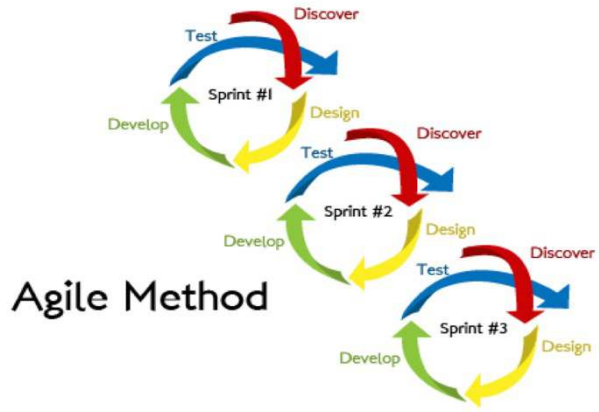


j. 10. Small release - turn over to user frequently
k. 11. Forty-hour work week - don't overload developers
l. 12. Coding standards - follow standards for code

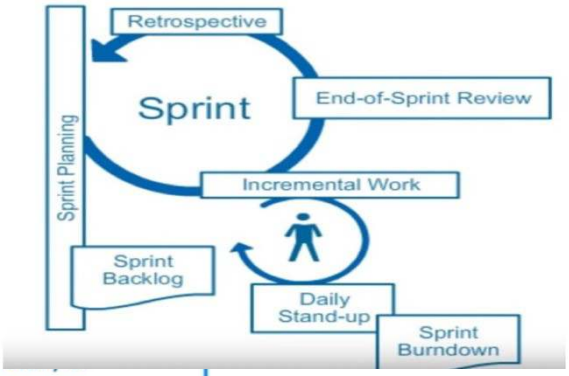
- <u>XP Project Approach</u>
  a. Three Levels
    - Outside ring - create user stories and define acceptance test
    - Middle ring - conduct test and do overall planning
    - Inside ring - iteration of coding and testing

<u>Scrum</u>

- Focus on incremental development, with each iteration all core processes of sys dev are repeated. More user involvement and tightly control schedules (use of small, self-organised work teams)
- Combo of principles of Rugby and Agile
- Product backlog
  a. Prioritized list of user requirements
- Product owners
  a. Client stakeholder who controls backlog
- Scrum master
  a. Scrum PM
- Scrum Sprint
  a. Time controlled mini-project to implement part of the system
  b. Few high-priority backlogs

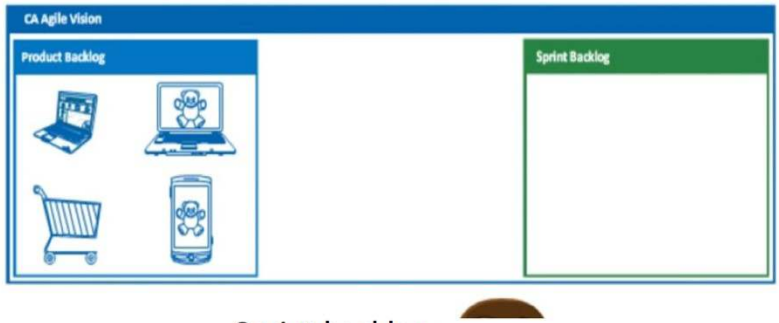


Agile Method

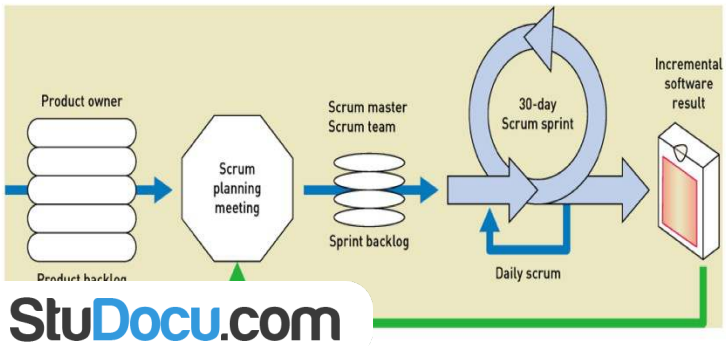


- **Scrum Practices**
  a. Scope of each sprint is frozen
  b. Time period is kept constant
  c. Daily scrum meeting
    - What have you done since the last daily scrum
    - What will you do by the next daily scrum
    - What kept you or is keeping you from completing your work
    - Progress reviews

Product backlog



Sprint backlog



Scrum life cycle

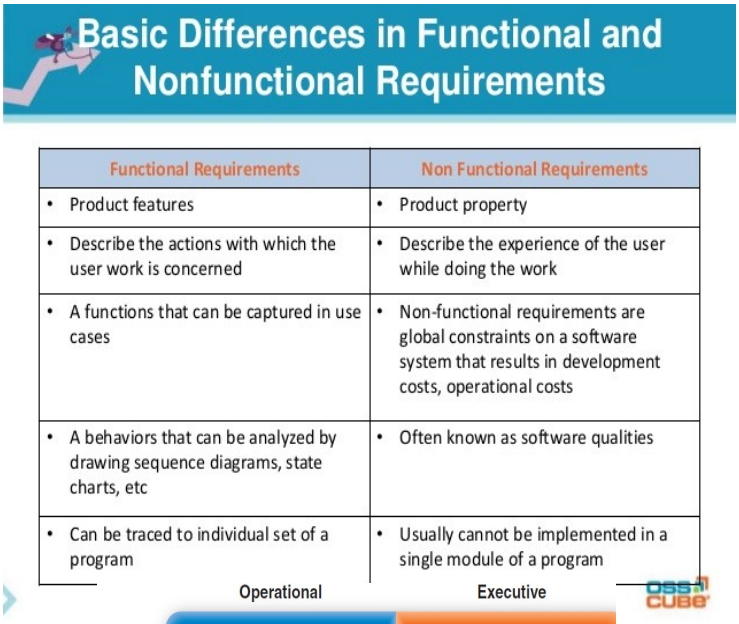
Week 3 – Investigating System Requirement

System Analysis Activities

- Gather Detailed Info
  - Interview, questions, documents, observing business process, research vendors, workshops
- Define Requirements
  - Modelling functional requirement and non-functional requirements
- Prioritize Requirements
  - Essential, important vs nice to have
- Develop User Interface Dialogs
  - Flow of interaction between user and system
- Evaluate requirements with users
  - User involvement, feedback, adapt to changes

What are requirements?

- System Requirements =
  - Functional
  - Non-functional
- Functional - activities the system must perform
  - Business uses, functions the user carry out
  - Shown as use case
- Non-functional - other system characteristics (quality property of functional requirement)
  - Constraints and performance goals



FRUPS + Requirements Acronym

- **F**unctional: tasks users perform using system, business rules/processes
- **U**sability: operational characteristics e.g. UI, colour choice
- **R**eliability: dependability, failure rate/recovery methods
- **P**erformance: response time, throughput
- **S**ecurity: password protection/encryption
- **+** even more categories

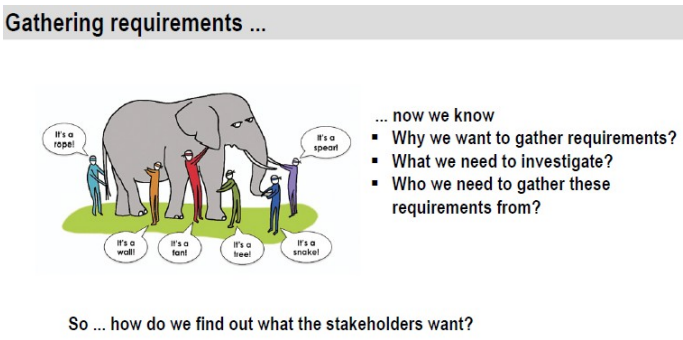
Additional Requirement Categories

- Design constraints
  - Specific restrictions for hardware and software
- Implementation requirements
  - Specific language, tools, protocols

- Interface requirements
  - Interface links to other systems
- Physical requirements
  - Physical facilities and equipment constraints
- Supportability requirements
  - Auto updates and enhancement methods

Stakeholder

- **Stakeholder -** person who have an interest in the successful implementation of the system
- **Internal Stakeholder -** person within the organization
- **External Stakeholder -** person outside the organization
- **Operational Stakeholder -** person who regularly interact with the system
- **Executive Stakeholder -** person who don't directly interact, but use the information or have financial interest



Information Gathering Techniques

- 1. Interviewing user and other stakeholders
- 2. Distributing and collecting questionnaires
- 3. Reviewing inputs, outputs and documentation
- 4. Observing and documenting business procedures
- 5. Researching vendor solutions
- 6. Collecting active user comments and suggestion

1. Interviewing Users and Other Stakeholders

- Prep detailed question
- Meet with individuals or groups of users
- Obtain and discuss answers to questions
- Document answers
- Follow up as needed in the future meetings or interviews

Theme	Questions to users
What are the business operations and processes?	What do you do?
How should those operations be performed?	How do you do it? What steps do you follow? How could they be done differently?
What information is needed to perform those operations?	What information do you use? What inputs do you use? What outputs do you produce?

Interview Follow up

- As documentation created after the interview should be reviewed for accuracy as soon as possible after the interview
- Follow up interviews are required to explain and verify the models with the interview participants, and ask further questions
- You will have unresolved issues
- They should be tracked and resolved

2. Distributing/collecting questionnaires

Channel: online, paper based, email
Advantage: cover wide spectrum of people
Disadvantage: low response rate

3. Review existing reports, forms and procedure descriptions
Internal business documents/procedure: be cautious of outdated documents and use as guidelines to guide interviews
External industry reports

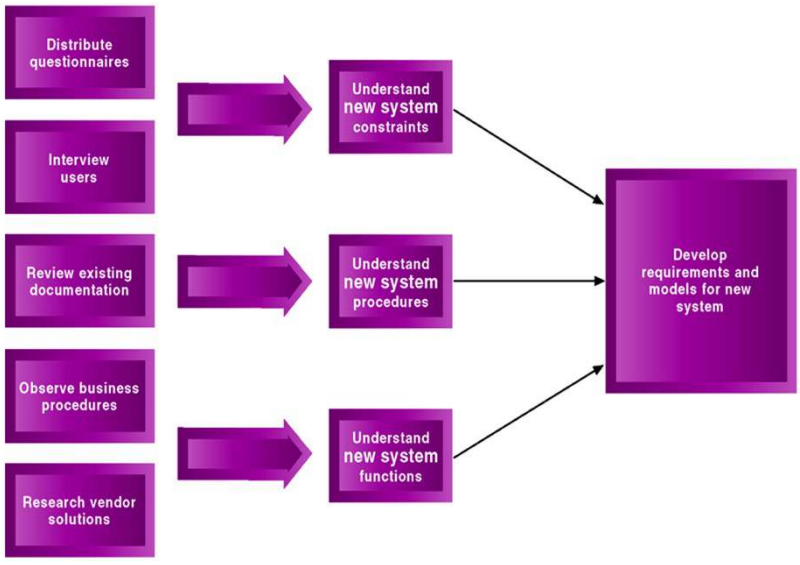
4. Observing/documenting business procedures
- Watch, observe and learn
- document with activity diagram, talk to frontline staff, see variations of official way and see how people interpret same data

Hawthorne effect: observer effect where workers improve/modify an aspect of their behaviour or activities OR stop working in response to the fact they are being observed

5.Research vendor solutions:
- business problems have already been solved by other companies, research similar situations
Positive: frequently provide new ideas, cheaper/less risky, get trial version
Risks: don't purchase solution before understanding problem, may not address all business requirements, vendor support may not exist in future, upgrade issues

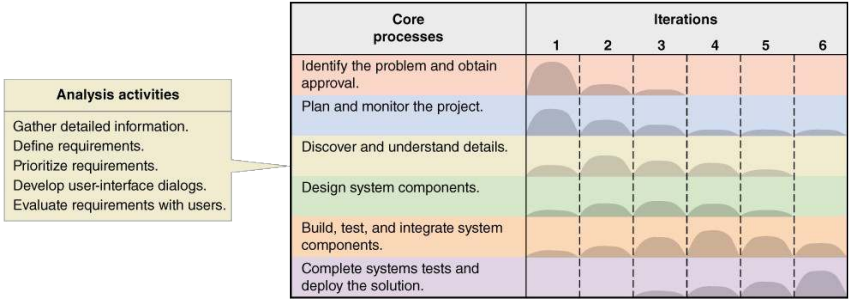
Workshops
- Sometimes referred to as JAD sessions:
  - Joint application development
- Get all stakeholder in a room for a couple of days and facilitate discussion
  - Build models with everyone
  - Share ideas
- Requires a conference room
- Can be difficult to organise
- Once initial session is complete, all notes must be written up into document
- This is then distributed for feedback

<u>Useful techniques in vendor research</u>
- Technical specifications from vendors
- Demo or trial system
- References of existing clients
- On-site visits
- Printout of screens and reports
- RFP - request for proposal



6.. Collect active user comments/suggestions
-feedback on models/test
(pic)

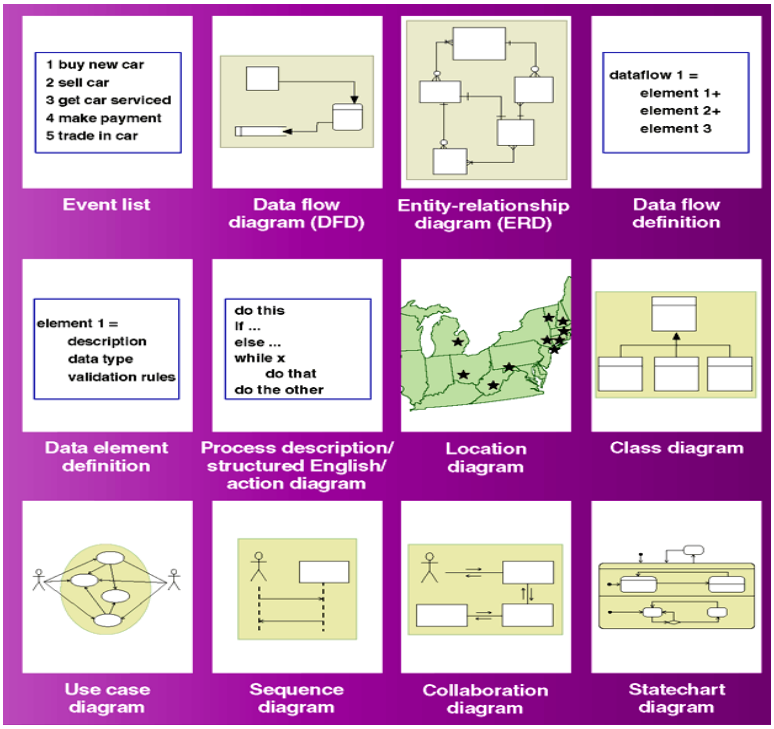
Models and Modelling
- Model – abstraction of the real item that is to be built (simplified picture of complex reality)
- Types of Models
  - Descriptive models - something written down – describe certain aspects of system
  - Physical models – scaled down version of real product
  - Graphical model - diagram, schematic – representations of some aspects of system
  - Math model - formula, stats, algorithms - describe technical aspects of system



- Unified Modelling Language (UML)
  - Standard graphical modelling symbols/ terminology used for IS

<u>Reason for Modelling</u>

- Learning from modelling process
- Reducing complexity by abstraction
- Remembering all the details
- Communicating with other development team members
- Communicating with a variety of users and stakeholders to clarify requirements
- Documenting what was done for future maintenance/enhancement
- Learning from modelling process



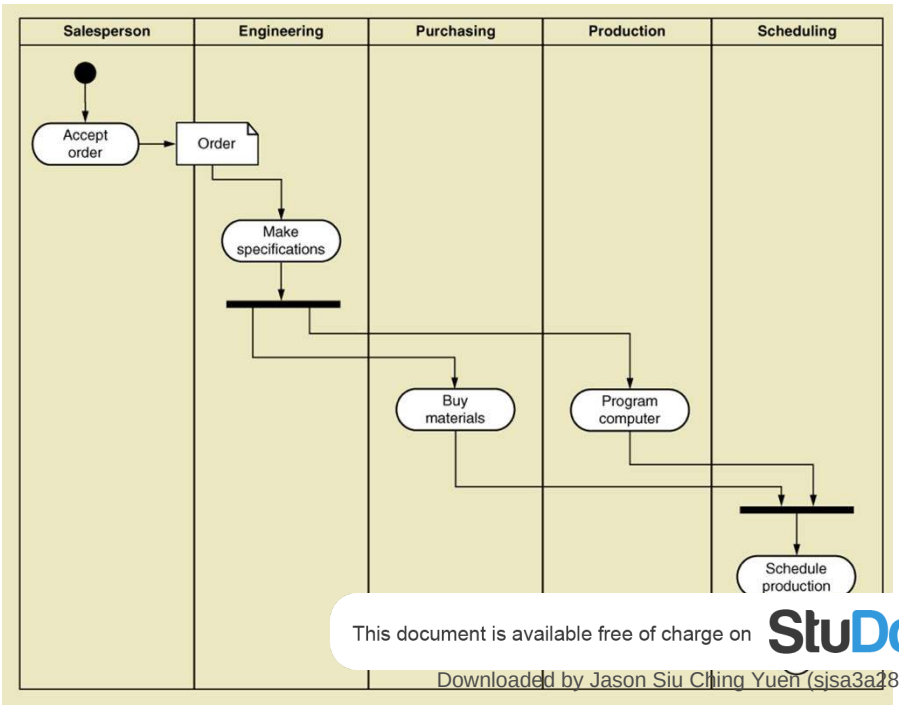
Types of models used in Systems analysis:
- logical models
- physical models
- dynamic models
- static models

Logical & Physical:
- describe WHAT IS DONE rather than HOW it's done
tech indep= provide details without referring to tech (Logical)
tech dep= explains how things are done (Physical)

Dynamic:
- express behaviour of system over time (time-dependent)
- represent object interactions during runtime (processes, workflows) e.g. Activity diagram models flow of control from activity to activity

Static:
- specifies structure of objects that exist in problem domain
- no time-dependency shown
- THINGS that business needs to store info e.g. Domain modelling/ER modelling

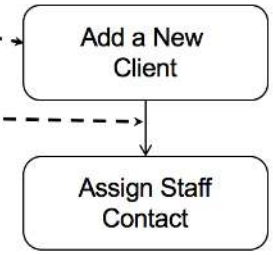
Documenting Workflow and Activity Diagrams



- **Workflow -** sequence of processing steps that completely handles one business transaction or customer request
- **Activity Diagram -** describes user/system activities, the person who does each activity, and the sequential flow of these activities
  - Useful for showing a graphical model of a workflow
  - A UML Diagram

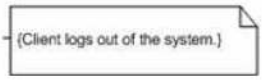
Diagram with concurrent paths

- Activity / Action
  - rectangle with rounded corners
  - meaningful name
- Control flows
  - arrows with open arrowheads (*direction of progressing activities*)

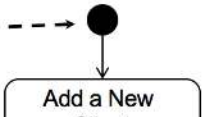


- Notes

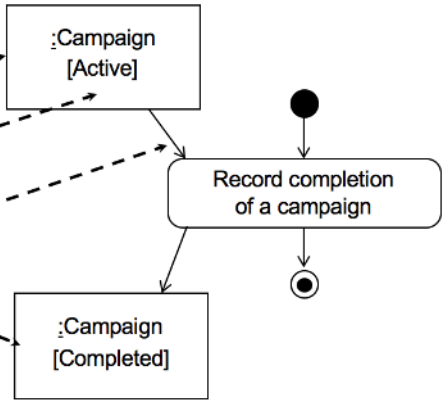
(*adds explanations*)



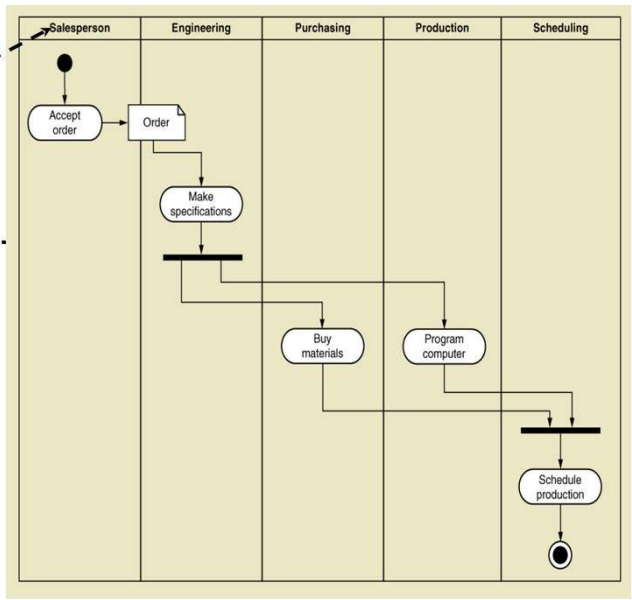
- Initial node
  - *black circle*



- Objects
  - Represents a business entity (e.g. document) or event that exists in a particular state
  - Shown as a rectangle
  - optionally shows *the state of the object in square brackets*
  - change state in an operation
- Object flows
  - open arrow
  - indicating which activity has affected the object-in-state



- Activity Partitions (**Swimlanes**)
  - vertical columns
  - labelled with the person, organisation, department or system *responsible for the activities* in that column

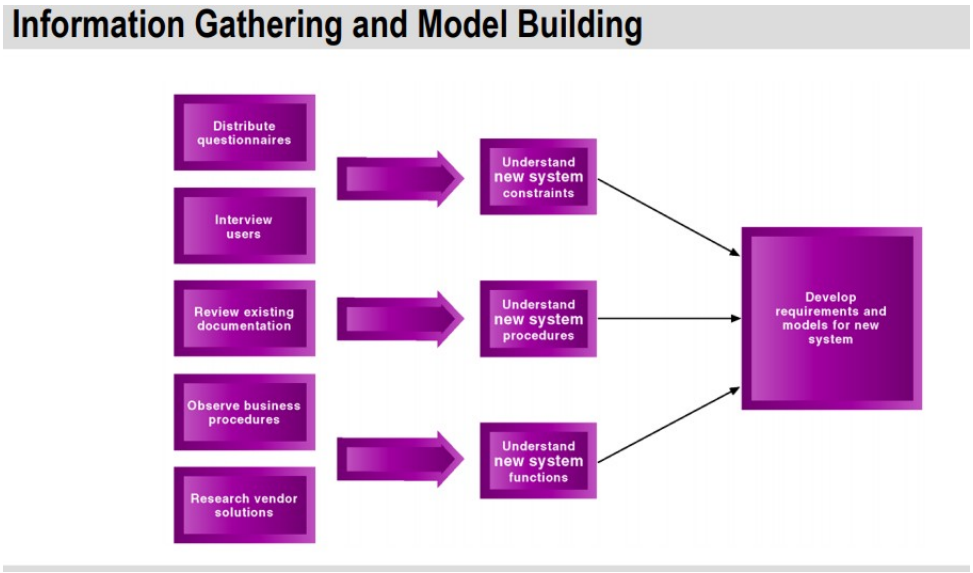


SUMMARY

- System analysis involves defining system requirements - functional and non-functional
- Analysis activities include;
  - Gather detailed info
  - Define requirements
  - Prioritize requirements
  - Develop user-interface design
  - Evaluate requirements with users
- FURPS + is acronym for functional, usability, reliability, performance, and security requirements
- Stakeholders are the people who have an interest in the success of the project
- Internal vs external and operational vs executive stakeholders
- Information gathering techniques are used to collect information about the project
- UML activity diagram is used to document (model) workflow after collecting information
- Models and modelling are used to explore and document requirements
- Unified Modelling Language is the standard set of notations and terminology for Information Systems Models

Week 4 – Identifying User Stories and Use Case

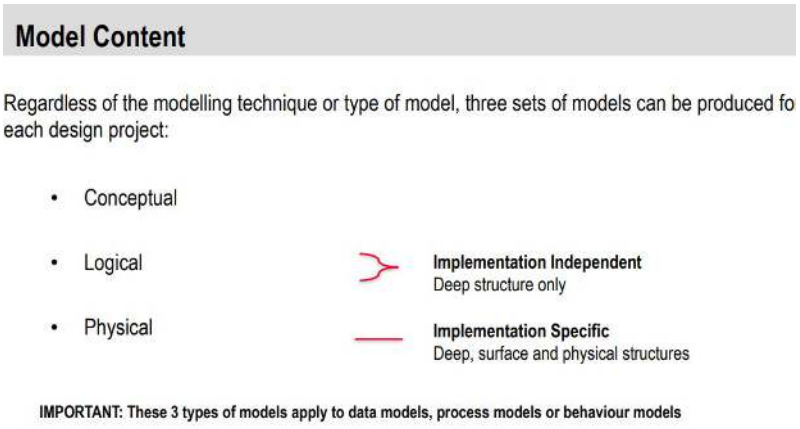
Identifying User Stories and Use Cases



Models – 3 different views of IS
1 – Processes/affordances (functions)
Relations in activities in processes by
- user stores
- use cases

2 – Data/info
relations in data
- class diagram
- ER Diagram

3 – Behaviour (combo of other two – OO models)



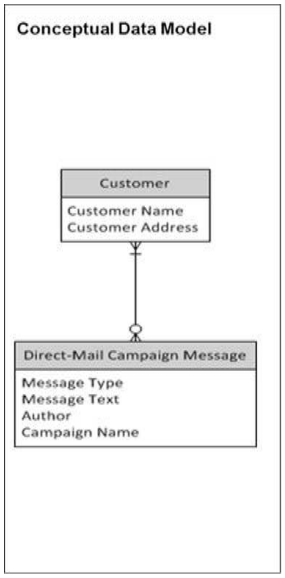
flow of tasks in processes
- activity diagram
- sequence diagram

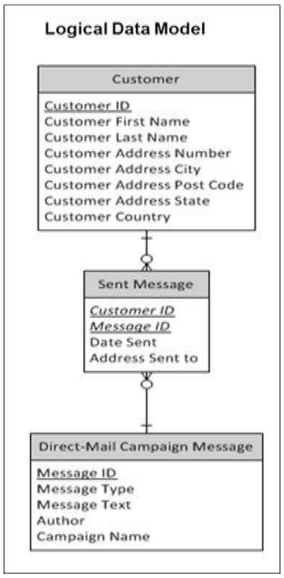
Conceptual	Logical	Physical
Conceptual model as a *first sketch of logical model*	Shows what a system will do. a *logical model as a detailed, complete*	Results from modifying the logical model to take *into account how the system will do what it is intended to do.*
technology independent	Still *technology independent*	A technology specific model.
Main audience: client, users, analysts	Main audience: client, users, analysts	Main audience: System developers and maintainers.
Conceptual model *allows analysts more freedom to communicate with clients - less complexity*	Protects business rules and requirements from technical, engineering considerations	Model is 'tuned' to take *into account performance requirements, efficiency* in processing, limitations of hardware, architectures

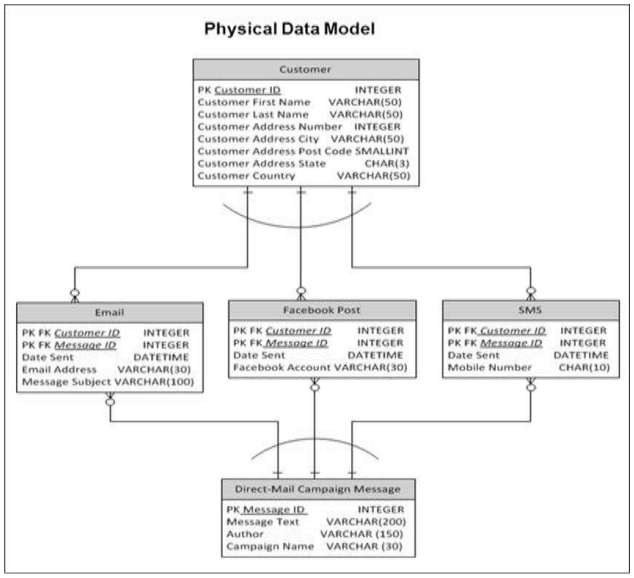
Conceptual Model
- Initial model developed with clients and subject matter experts
- Business oriented
- Most creative, design oriented model
- Not constrained by anything other than most basic notations
- Aims to specific what the system should do, independent of how it does
- Main audience: client, users and experts

Logical Model
- Result of modifying conceptual model
  - Complete, detailed specification of functionality and info structure
  - Some designers treat conceptual and logical as same thing
- Still tech independent
- Think conceptual as first draft, logical as detailed, complete and technically correct final draft
- Main Audience: client, users, experts, developers, designers

Physical Model
- Technology specific model
- Results from modifying the logical model to take into account how the system will do what it is intending to do
- Technically oriented
- Models surface and physical structure as well as the deep structure
- Model is tuned to take into account performance requirements, efficiency in processing, limitations of hardware, architecture
- Main Audience: System Development and Maintainers







Why we do this?

- Having separate logical and physical models allows us to fine tune the implementation, upgrade components, without impacting the fundamental design of the system
- Protects business rules and requirements from technical, engineering considerations
- Conceptual models allow us more freedom to communicate with clients - less complexity

Who should be involved

- Designer
- System owner, user, project sponsor
- Subject matter experts
- IT department

USER STORIES

- Is a one sentence description of a work-related task done by a user to achieve some goal or result
- Acceptance Criteria identify the features that must be present at the completion of the task
- Describe user requirements and document functional requirements
- ACCEPTANCE CRITERIA - The template for a user story description is
  - AS A <ROLE> I WANT TO <GOAL> SO THAT <BENEFIT>

USE CASE

- Use case - an activity that the system performs, usually in response to a request by a user
- Use case defines functional requirements
- Involves list of steps between a role (actor) and system, to achieve goal
- Analysts decompose the system into a set of use cases (functional description)
- Expressed in 3 formats: tabular format w/o details, tabular format with details, and graphical format
- Two techniques on HOW to identify use cases
  - User goal techniques: goals for interacting with new system
  - Event decomposition techniques: business events identified
- Name each use case using **verb-noun**

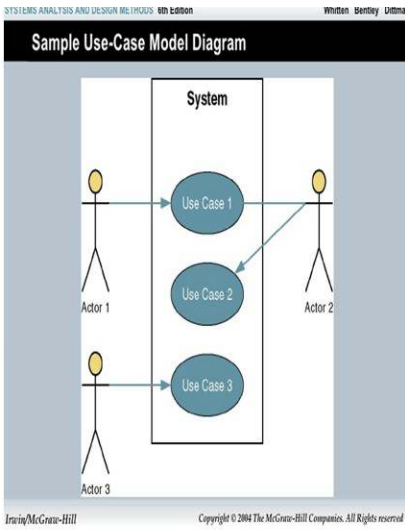
Similarity (Use Case/Stories)	• **They both focus on the goals of users** • **They both describe functions that users want**
Differences	• **The degree of details differ**

- Use cases are expressed in **THREE formats**
  - Tabular format without details(verb-Noun)
  - Tabular format with details (Narrative format)
  - Graphical format

User	User goal and resulting use case
Potential customer	Search for item Fill shopping cart View product rating and comments
Marketing manager	Add/update product information Add/update promotion Produce sales history report
Shipping personnel	Ship items Track shipment Create item return

Name each use case using Verb-Noun: Examples shown above

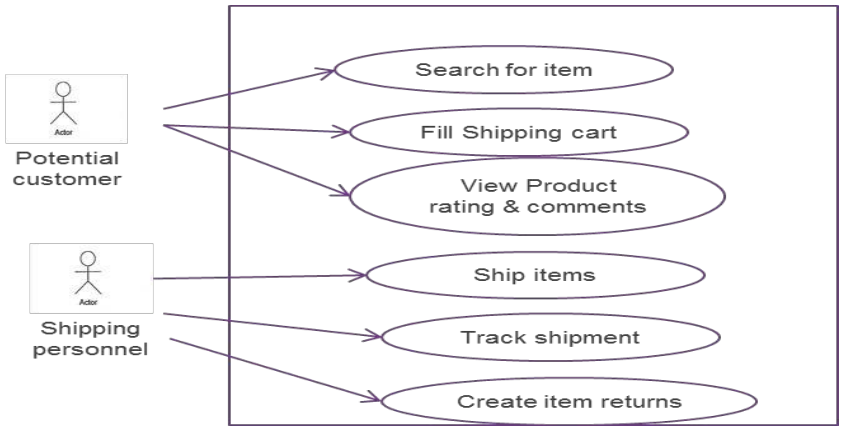
Use Case Template	'Create Order' Use Case
Use Case Name	Create order
Use Case Description	Create order is the ability to request the purchase of a product
Actor	Order Creator
Pre-conditions	• Order Creator has been identified
Basic Flow	1. Order Creator selects 'order product' action 2. System requests customer/product identification information 3. Order Creator provides customer/product identification information 4. System requests mailing information 5. Order Creator provides mailing information 6. System verifies mailing information 7. System requests order be submitted 8. Order Creator submits order 9. System submits product order for processing 10. System confirms product order
Post-conditions	• Product order has been created
Alternate Flows	• Product is not in stock • Product has been discontinued • A customer's initial order is over $200



A **sample use case diagram** is shown which includes 3 use cases and 3 actors (roles)

1. User Goal techniques

- Identify all the potential categories of user of the system
- Interview and ask them to describe the tasks the computer can help them with
- Probe further to refine tasks into specific user groups
  - I need to Ship items, track a shipment, create a return

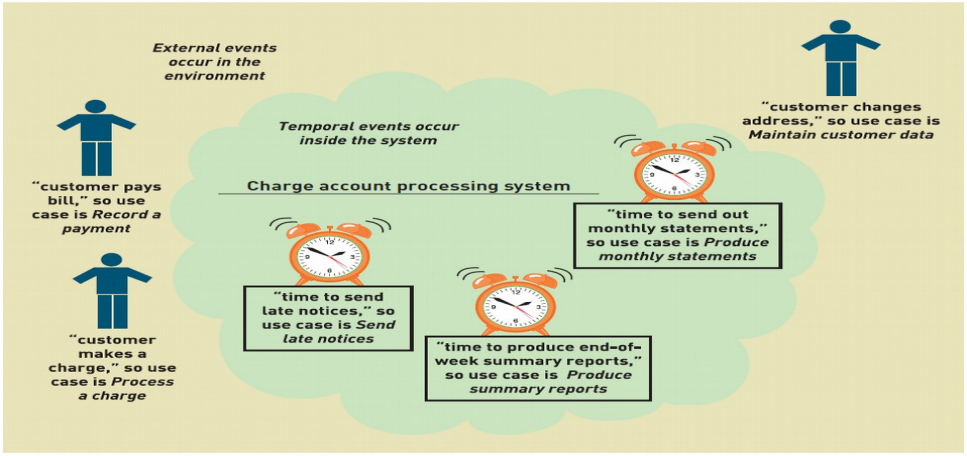


User Goals Technique: Specific Steps

1. Identify all the potential users for the new system
2. Classify the potential users in terms of their functional role (sales, shipping)
3. Further classify potential users by organizational level (executive, management)
4. For each type of user, interview them to find a list of specific goals they will have when using new systems
5. Create a list of preliminary use cases organized by type of user
6. Look for duplicates with similar use case names and resolve inconsistencies
7. Identify where different types of users need the same user cases
8. Review the completed list with each type of user and then with interested stakeholders

2. Event Decomposition technique

- More comprehensive and complete technique
  - Identify the events that occur to which the system must respond
  - For each event, name a use case (verb-noun) that describes what the system does when the event occurs
- Event something that occurs at a specific time and place, can be described and should be remembered by the system
- Types of events: external, temporal and state

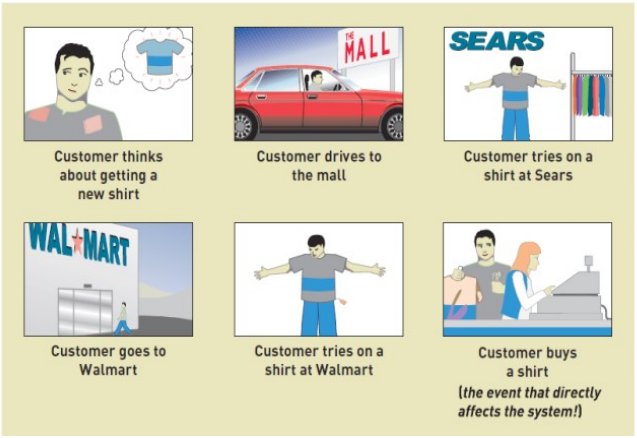


Types of Events

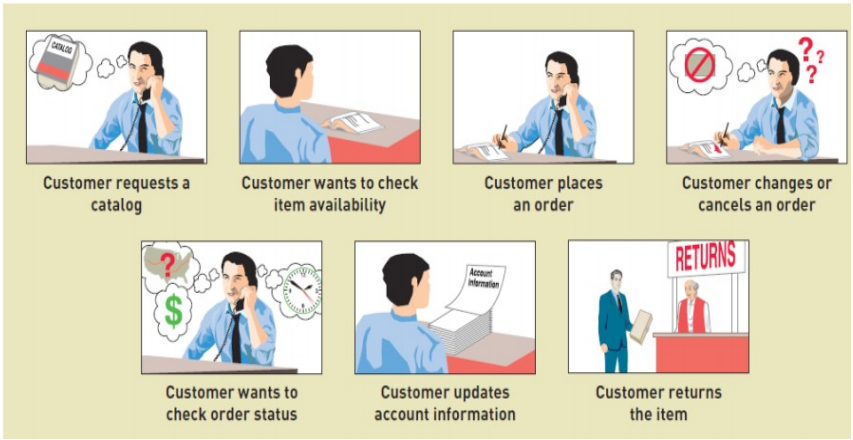
- External events
  - An event that occurs outside the system, usually initiated by an external agent or actor
- Temporal Event
  - An event that occurs as a result of reaching a point in time e.g. monthly bills
- State Event
  - An event that occurs when something happens inside the system that triggers some process

E.g. Reorder point is reached for inventory item

Finding the actual event that affects the system



Tracing a sequence of transactions resulting in many events



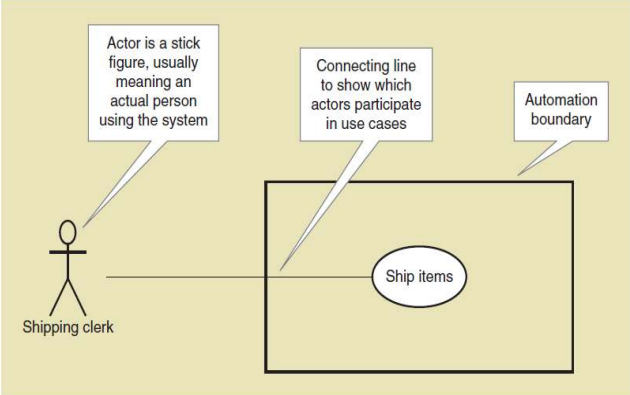
E.g. customer at counter paying to buy shirt called EBP: Elementary business process
Use cases should be defined at EBP process level. It is a task performed by one person in 1 place in response to business event and leaves the system/and data in stable state

Event Decomposition techniques: Benefits

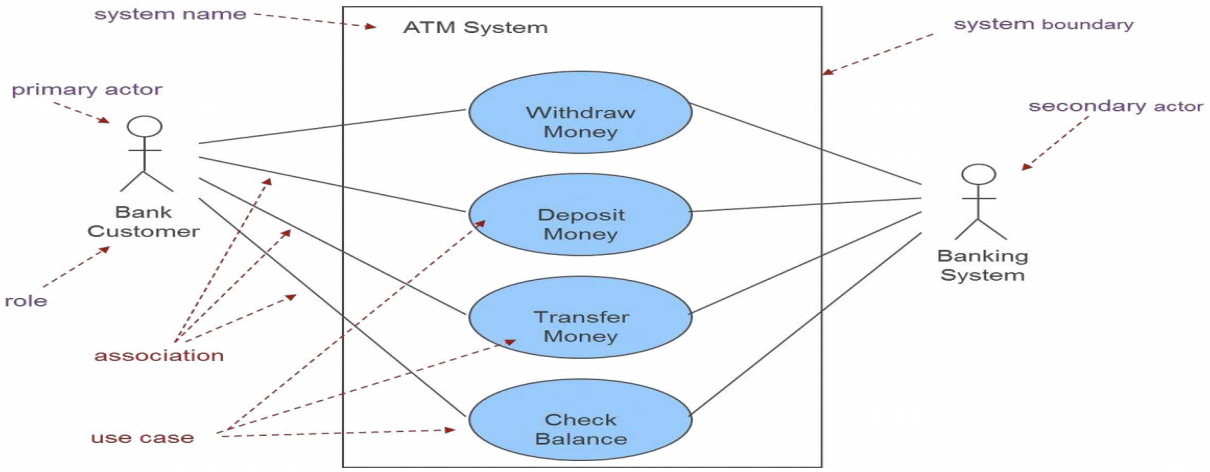
- Events are broader than user goals: Capture temporal and state events
- Help decompose at the right level of analysis: EBP
- EBP is a fundamental business process performed by one person, in one place, in response to a business event
- Uses perfect technology assumption to make sure functions that support the users work are identified and not additional functions for security and system controls

Use Case Diagram

- A UML model used to graphically show uses cases and their relationship to actors
- Recall UML is Unified Modelling Language, the standard for diagrams and terminology for developing info systems
- Actor is the UML name for end user. An actor is a person/entity that initiates functionality provided by a Use Case
- **Automation Boundary -** the boundary between the computerized portion of the application and the users who operates the application
- User may have more than one role**:** Primary, Secondary
- **Primary actors:** actors using system to achieve a goal. Documents interactions between system and actors to achieve goal of primary actor
  Secondary actor: actors that the system needs assistance from to achieve the primary actor's goal



- Association line indicates that a particular actor makes use of the functionality provided by particular use case
  System boundary defines scope of what system will be.



Relationship between 2 use cases is essentially a dependency between 2 use cases
<<Include>> - captures commonality among use cases, reusability (saves time/money)
<<Extend>> optional use case executed when conditions are satisfied (parent case can function without child case) Tip of arrowhead points to parent use case and child use case connected at base.

Use Case Diagram: Steps

1. Identify all stakeholders and users who would benefit by seeing a use case diagram
2. Determine what each stakeholder or user needs to review in a use case diagram: each subsystem, for each type of user, for use cases that are of interest
3. For each potential communication need, select the use cases and actors to show and draw the use case diagram.
4. Carefully name each use case diagram and then note how and when the diagram should be used to review use cases with stakeholders and users

- Use case ranking/priority matrix:
  Evaluates use cases on 1-5 scale against six criteria.
  1. Significant impact on the architectural design.
  2. Easy to implement but contains significant functionality.
  3. Includes risky, time-critical, or complex functions.
  4. Involves significant research or new or risky technology.
  5. Includes primary business functions.
  6. Will increase revenue or decrease costs.

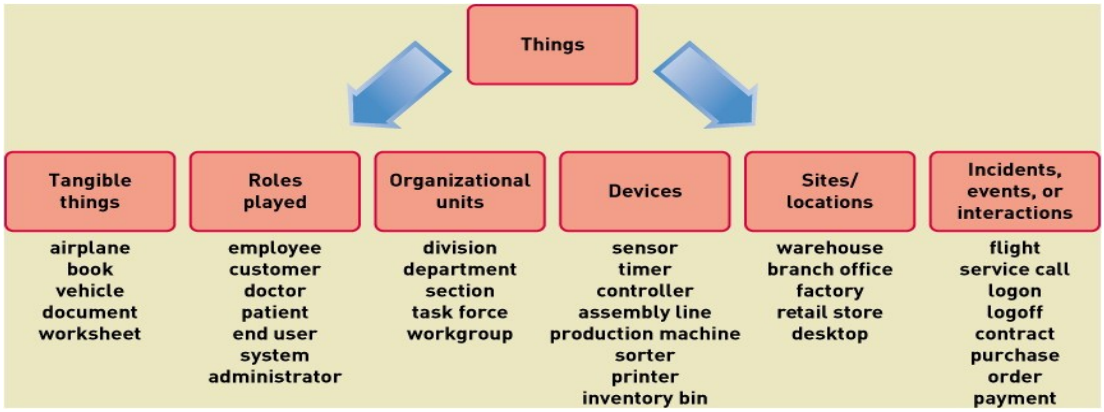
SUMMARY

- Use cases are the functions identified, the activities the system carries out in response to a user request
- Two techniques for identifying use cases are the user goal technique and the event decomposition technique
- The user goal technique begins with identifying end users called actors and asking what specific goals they have when interacting with the system
- The event decomposition technique begins by identifying events that occur that require the system to respond
- Three types of events include external, temporal and state events
- Brief use case description are written for use cases
- The use case diagram is the UML diagram to show the use cases and the actors
- The use case diagram shows the actors, automation boundary, the use cases that involves each actor, and the <<includes>> & <<excludes>>relationship
- A variety of use case diagram are drawn depending on the presentation needs of the analysis

Week 5 – Domain Modelling

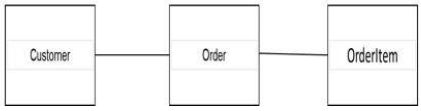
Domain modelling:
- a live collaborative artefact (between users/analysts), refined and updated through the systems development (through iterations)
- THINGS = classes in UML
- represent a graphical way of representation of relationships between things (classes) in system that analysts need to keep info about in the business problem domain
- static in nature
- tangible vs intangible
- THINGS & CRUD

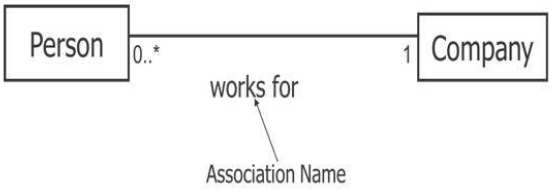
1. Identify user and set of use cases
2. brainstorm THINGS involved when carrying out a use case
3. Ask questions



- Once things are identified, shown as classes of objects as a rectangle (single domain class)

  - consider relationships between the domain classes (entity relationship)





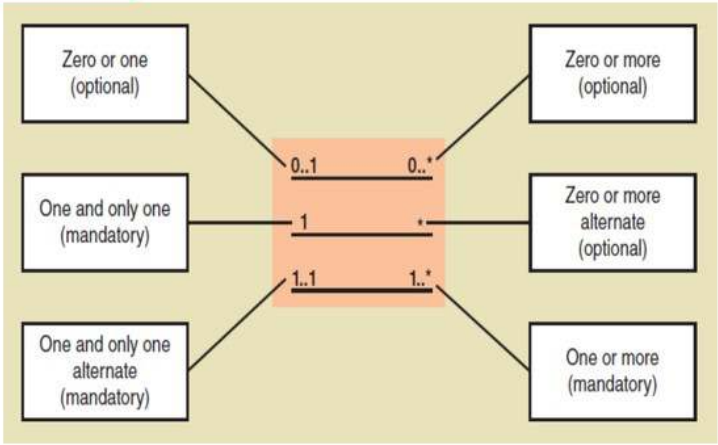
- A customer places orders
- An order has items

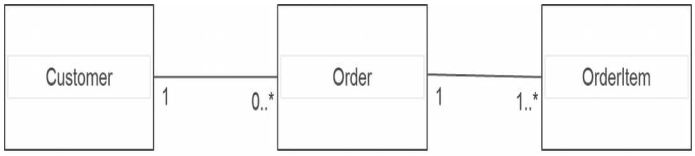
- Multiplicity: It refers to *the number of association* between 'things' (classes) – established *for each side of an association* .. *in ER modelling known as Cardinality*

Minimum and Maximum multiplicity

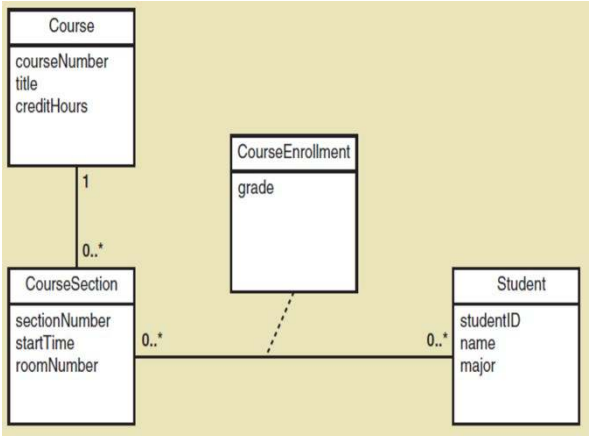
- When, minimum is zero, the association is OPTIONAL
- If minimum is at least one, the association is MANDATORY

Multiplicity Indicator	Meaning
0..1	Zero or one
1	One only
0..* or *	Zero or more
1..*	One or more
3	Three only
0..5	Zero to Five
5..15	Five to Fifteen



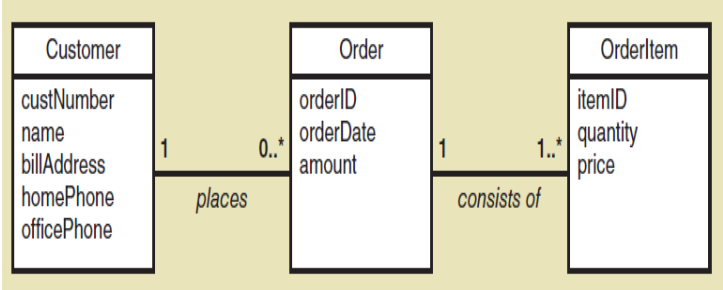


- A customer may have placed no orders, or they could have placed many orders
- An order can only have been placed by a single customer
- An order must consist of at least one item but could consist of many items
- An ordered item can only belong to one order



Association class: treated as class in a many-many association because it has attributes that need to be remembered e.g. Grade

Attribute: specific piece of info about a THING
(have value for each of these)
Compound attribute: set of related attributes



Generalisation/specialisation relationships

hierarchical:
subordinate class are special types of the superior classes
- Inheritance hierarchy
"Is-A" type of relationship where subclasses inherit characteristics of more general superclass

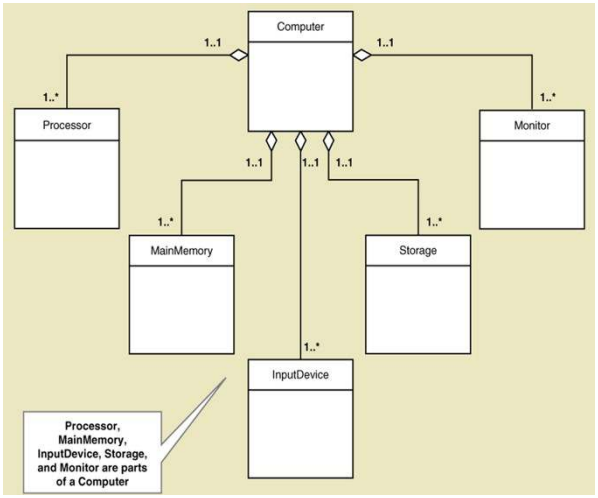
superclass:
superior/more general class

subclass: subordinate or more specialised class

inheritance: concept that subclasses inherit characteristics of more general superclasses

Abstract class: class that allows subclasses to inherit characteristics but never get instantiated
Concrete class: class that can have instances

Whole–part relationship:
refers to type of relo between classes where 1 class is part/ component portion of another class
"Has-a" relationship
2 types of whole-part associations: aggregation, composition



Aggregation: models whole-part relo between aggregate (whole) and its parts
component part can exist separately, can be removed/replaced
Unfilled diamond used

Composition:
form of aggregation with strong ownership, strong lifecycle dependency between instances of container class and instances of contained classes (filled diamond)

Summary

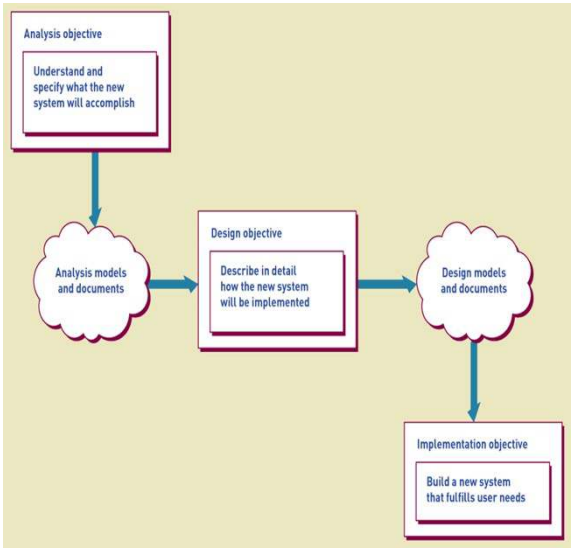
- Two techniques for identifying domain classes/data entities: brainstorming and noun
- Domain classes have attributes and associations
- Associations are naturally occurring relationships among classes, and associations have minimum and maximum multiplicity
- The domain model classes do not have methods because they are not yet software (design) classes.
- There are actually three UML class diagram relationships: association relationships, generalisation/specialisation (inheritance) relationships, and whole part relationships
- Abstract versus concrete classes

Week 6 – System Design Fundamentals

Systems Design Fundamentals

What is System Design?

- Analysis provides the starting for design
- Design provides the starting point for implementation
- Analysis and design results are documented to coordinate work
- Objective of design is to define, organize, and structure the component of the final solution to serve as a blue print for construction
- Focuses on HOW software systems built, the structural components and shows blueprint for software construction design represents bridge from requirements to actual implementation



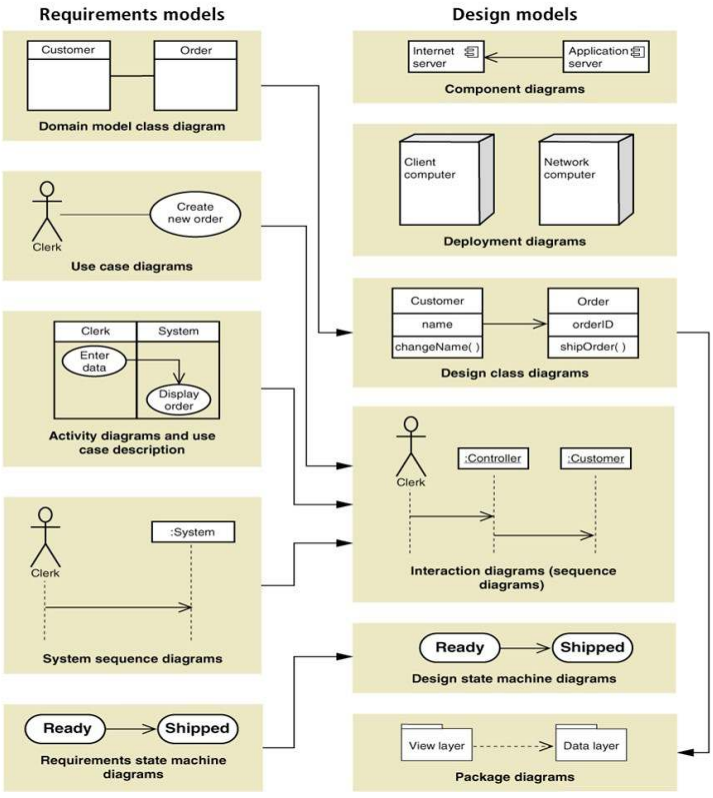
2 levels:
Architectural design vs Detailed design

Architectural:
- high level design of overall system structure (Conceptual design)

Detailed design
- design of specific details; use case, database, user/system interfaces, controls/security

Design Models

- Design is a model building activity
- The formality of the project will dictate the type, complexity and depth of models
- Agile/iteration projects typically build fewer models, but models are still created
- Jumping to programming without design often causes less than optimum solution and may require network



Design Activity

- DA correspond to components of the new system (all 6 SDLC processes involved in agile/iterative development)
  - activities can be done in parallel or influence each other greatly

(5 major Design system components)

- The environment
- Application components
- User interface
- Database
- Software classes and methods

Design activity	Key question
Describe the environment	How will this system interact with other systems and with the organization's existing technologies?
Design the application components	What are the key parts of the information system and how will they interact when the system is deployed?
Design the user interface	How will users interact with the information system?
Design the database	How will data be captured, structured, and stored for later use by the information system?
Design the software classes and methods	What internal structure for each application component will ensure efficient construction, rapid deployment, and reliable

1. **Describe the Environment**

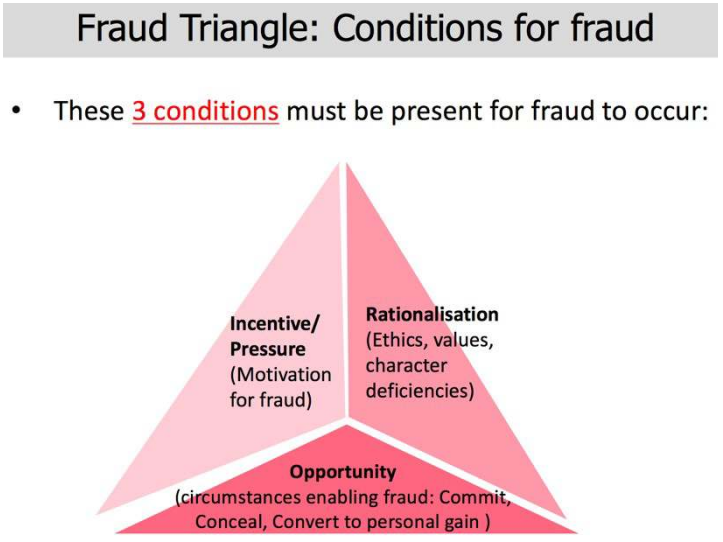
- Two key elements in the environment
  - Communication with external system
    - Message formats
    - Web and networks
    - Communication protocols
    - Security methods
    - Error detection and recovery
  - Conforming to an existing Technology Architecture
    - Discover and describe existing architecture

2. **Design the Application Components**

- App component is a well-defined unit of software that performs some functions
- Issues involve how to package components including
  - Scope and size - What are the functions, boundaries, interfaces?
  - Programming language - What are the accepted languages?
  - Build or buy - Is an acceptable version available to purchase?
- 4 design components:
  1- server (software applications/databases)
  2- network (how application is deployed connecting clients)
  3- controls (integrity i/o, security e.g. firewall)
  4- UI design
- **Controls:**
  Input – prevent invalid/erroneous data from entering system
  Value limit controls: check range of inputs for reasonableness
  Completeness controls
  Data validation controls
  Field combination controls
  Output – ensure output arrives at proper destination, accurate, current and complete

Fraud prevention – internal fraud/embezzlement/loss

Fraud Triangle:
Incentive/pressure: Financial/emotional/lifestyle determines incentive to commit fraud.
Opportunities: conditions to commit fraud, conceal and convert fraud – e.g. selling in black market
Rationalise: How to justify the fraud, lack of integrity or moral values



Security controls:
- protect assets against external threats
- maintain stable, functioning operating environment always
- protect info/transactions (data encryption)
encryption- alter data so its unrecognisable
decryption- convert encrypted data to readable form
encryption algorithm- mathematical transformation of data
encryption key- long data string allows the algorithm to produce unique encryptions
cryptography: mathematical process of encrypting/decrypting message (symmetric and asymmetric)
symmetric key encryption- (private key-known to user) uses same key to encrypt/decrypt
asymmetric- (public key-known to anyone) two separate keys used which are mathematically dependent on each other
public key applied via-
digital signature: document encrypted using private key, decrypt using correct public key

use of the keys required trusted org to act as Certification Authority which:
- issue public/private keys
- maintain listing of users and public keys (directory of public keys)

3. **Design the User Interface**

- To the user, the user interface is the whole system
- The user interface has large impact of user productivity
- Includes both analysis and design tasks
  - Requires heavy user involvement
- Current needs require multiple user interfaces
  - Many different devices and environment, for different types of end users

4. **Design the Database**

- Requires converting the data model to a rational database
- Frequently use Relational database management system (RDBMS)
- Requires addressing of many other technical issues
  - Throughput and response time
  - Security

5. **Design Software Classes and Methods**

- AKA detailed Design
- Model building activity
  - Design class diagram
  - Sequence diagram
  - State-machine Diagram

Describe the environment:

Deployment environment represents infrastructure on which all software operates (hardware, system software, O/S, databases, firewalls etc.) with many variations available

Major trends-

1) internal deployment (stand-alone systems, two tier LAN systems, three layer client server architecture)

Standalone – run on one device w/o networking
data backup time consuming, data security
Two-tier – servers/clients
Three-layer – client, domain layer, data layer (middle layer)
middle layer – translates client request into data access commands, called application logic layer
(reduce workload of data servers, relieves clients of complex tasks)

View Layer: client computers and application server, formats HTML is application server
Data layer: database server and app to access data
Business logic layer: application server and includes logic to process business rules

2) external deployment

hosting alternatives for internet deployment: virtual servers, cloud computing
diversity of client devices: full size, tablets/notebooks, mobile phones
- presence of firewall, clients outside organisation

firewalls: software/hardware based
aims to shield computer/network that permit legitimate communications to pass and stops unauthorised connections, regulate traffic

- performance improved by multiple server configurations but complexity increases

- Hosting (data centres)
operating data centre on someone's behalf where application software and database reside
- providing physical servers at secure locations, selling those services to other businesses that wish to deploy websites

3) remote, distributed environment

VPN – virtual private network
Closed network with security/closed access built on top of public network (internet)
e.g. interfaces for internal vs. external access

Design system interfaces – (more integration now, interacts with other systems)
save, read data that another system can use and request real time info

BUILD VS BUY

Build (in-house dev)
- assemble internal project team to develop, maintain, upgrade and meet requirements of business
- takes greater time to develop, test and upgrade due to training and customisation

Buy
- commercially available, standardised and bought from software vendors, can customise slightly (value added reseller - VAR), can be used across multiple orgs, vertical application (developed to meet a particular type of business), can have both

Reasons to buy:
- needed for standard business function e.g. payroll
- resources short in supply
- more cost effective
- needed in short time
- maintained by vendor, no internal development is necessary
- upgrades done by vendor
- high quality

Limitations to buy:
- can't meet all requirements
- need customisations, vendors can refuse to support customised packages after, and can be so extensive its cheaper to develop in-house
- integration costs

Package selection steps:
1- identify products that match requirements
2- solicit, evaluate and rank vendor proposals
3- select best vendor
4- integrate vendor product with existing system

Week 7 – Designing the UI

UI represents set of I/O that directly involve users of a system – what users see/interact
- represent one of the 5 key activities involved in systems design of SDLC
- usability improved by good UI, essential to promote acceptance of system

end-user= entire system shown to them (what's behind is irrelevant)

Good UI requires:
- proactive involvement of end users
- rich interactions b/w system designers/end user
Complexity increased coz:
- users have multiple devices
- they use different O/S
- purpose
- user characteristics

Physical aspects: touched by user
Perceptual aspects: hears, sees
Conceptual aspects: what users know about system/logical functions

user experience (UX) is a broad concept that applies to all aspects on end users interactions with an IT system
includes: action, responses, perceptions & feelings

User centred design:
- focus early on users and their work (stakeholders identified and their needs)
- evaluate designs to ensure usability
- use iterative development

use of metaphors – UI and aspects of physical entity, helps reduce cognitive load

PRINCIPLES: (6)
1. Human interface objects
2. Consistency
3. Discoverability
4. Closure
5. Readability/navigation
6. Usability/efficiency

1. Human interface objects =
objects users can manipulate
affordance= appearance suggests function
visible with feedback= gives response to user action (e.g. play button) – primary and secondary action buttons

2. Consistency:
- across platforms, within application and suite of applications

3. Discoverability
don't clutter UI

4. Closure
should know when task completed, provide undo/reverse actions

5. Readability/navigation
- identify needs of target audience, and implement readable text for all users

6. Usability/efficiency
- shortcut keys for experienced users, meaningful error messages, simplicity, KISS (keep it simple stupid)

FACTORS AFFECTING DESIGNING OF UI =
1. Type of system
2. Device
3. What O/S UI will function on

Issues:
1 – layout/format
guideline 1 – single primary purpose per screen
guideline 2 – screen layout top to bottom and left to right
guideline 3 – align/organise screen objects and grammatical errors

data entry, navigation/visibility

foundation of UI built on: use case, activity diagrams and system sequence diagrams
menu design = group functions so users easily locate, consider organisation, number of choices and limits of human cognition
CRUD – into a single menu

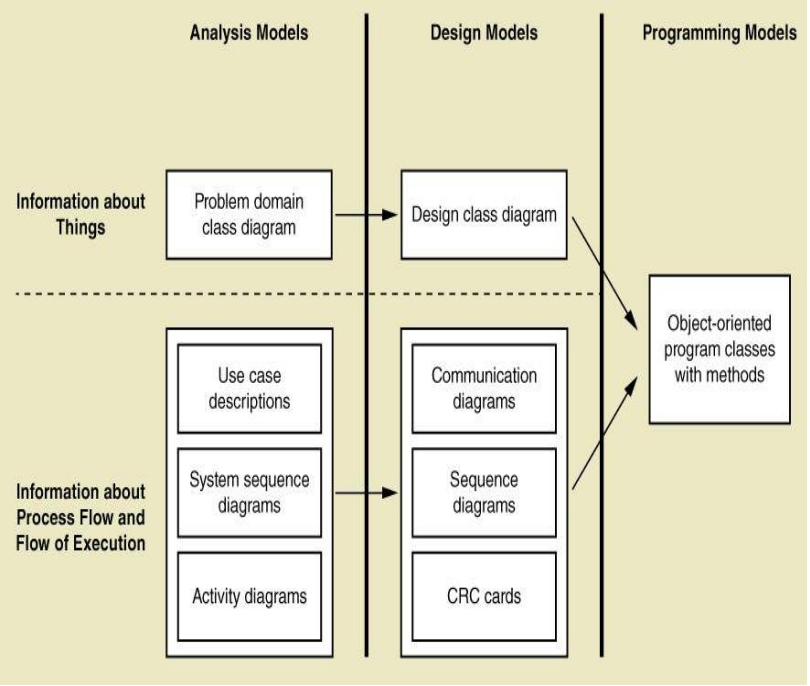
storyboarding = technique that shows sequence of sketches of display screen during dialogue, not very detailed, review with users and can be represented with e.g. powerpoint
UI Design for smartphones is hard coz small screen size, small keyboards, touch screen, limited network capacity
Issues: layout/formatting, data entry/user action, navigation/visibility

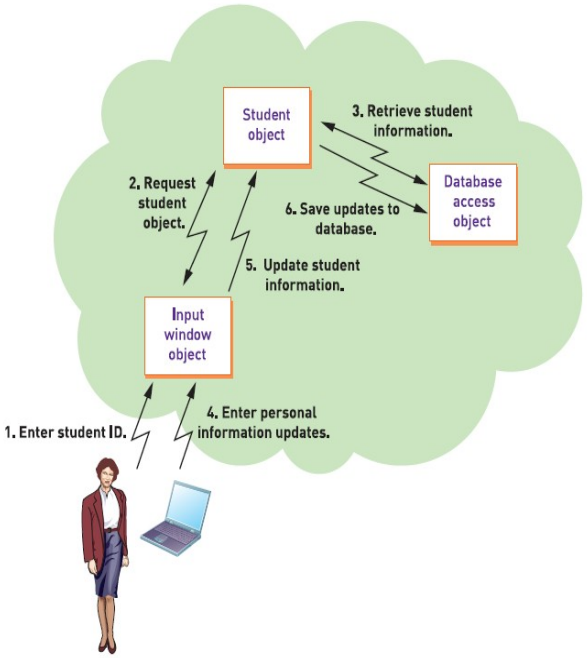
Designing reports:
screen views: print view, printed report
factors affecting info on report: manager needs, and how to display those need
types: detailed reports, summary reports, exception reports, executive reports, graphical/multimedia report

SUMMARY

- UI involves direct user interaction with the system
- Design of UI has long history as HCI and relies on user-centred design, which focuses early on users, evaluated design to ensure usability and uses iterative development
- Metaphors are used to think about the nature of the UI, and they include direct, desktop, document and dialog
- Key UI concepts include affordance and visibility for controls
- Other key principles include consistency, shortcuts, feedbacks, dialog closure, error handling, and reversal of action
- Use cases are organized into one or more menu hierarchies to arrange functionality for users
- Dialog and storyboard are used to design the interaction for each use case based on use case flow of activities and SSD
- Guidelines are available for designing Windows, Web Browser and handheld devices
- Designing inputs involves identifying devices and mechanisms, identifying inputs and the data content, and determining the controls necessary
- Designing outputs includes designing detailed reports, summary report, exception report and executive reports
- Electronic reports and other outputs can include drill down, graphic and multimedia

Week 8 – Object-Oriented Design and UML Modelling (Part 1)





Class = thing (tangible/visible) that people/systems interact with, info needs to be stored and easily identified. It is a design concept but can have multiple objects created at runtime (e.g. new Class). It's a collection of the same type of objects with properties/methods

Object = instance of class during runtime. They perform operations defined by a class.
Objects communicate to another object by passing messages (attributes, methods performed, what other objects it is associated and hence represents

a self-contained unit)

An object belongs to a single class – all objects within class share common attributes which may have different attribute values

domain class model: conceptual/logical model of "thing" – displays associations between things and business rules
design class model: physical model of objects of a class (things) add in tech attributes/classes – controller class

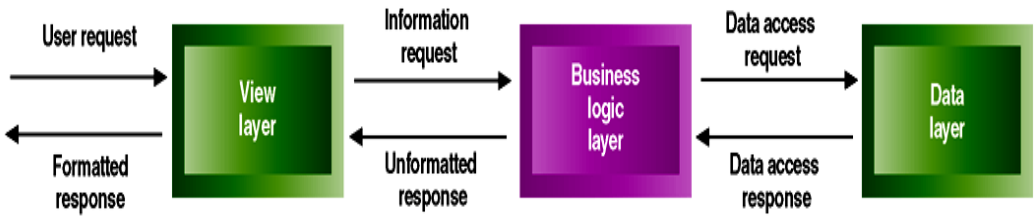
OO design: process by which a set of detailed OO design models are built for program coding. (program code based on design model spec)
Analysis models: use case, sequence diagrams, class diagrams are extended to design models (created in parallel to actual coding with iterative SDLC)

OOD provides blueprint to write code via design class diagram/sequence diagram.
OOP contains objects (from classes) with attributes, communicates messages via methods

Input window object – view layers which accepts user inputs and formats displays processing results

Business logic or domain layer (student object)– which implements the rules and procedures of business processing

The data layer (DB access object) - which manages stored data usually in one or more databases – and communicates with domain layer
(may physically reside on 3 servers)



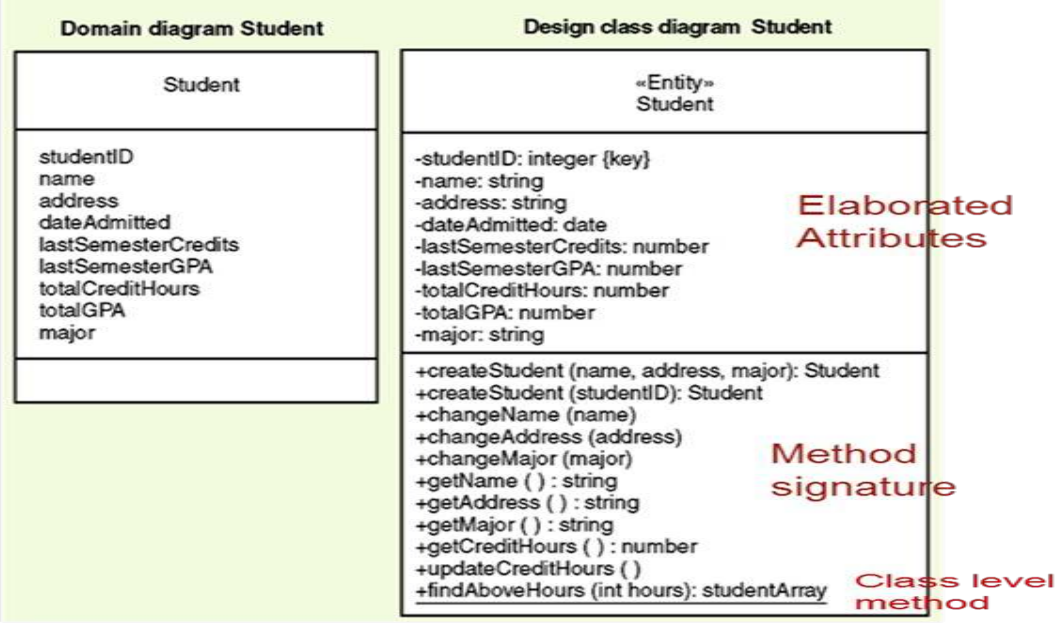
Design models:
System seq diagram (SSD): high level interaction between actor and system
can be expanded to show interaction b/w 2 internal objects

OOP has access to classes in 3 layers:
UI, problem domain and database access

OOD issues:
1) OOD represents process that identifies classes, methods and the messages with each layer
2) OOD is use case driven. Single use case chosen with appropriate models which are used to describe details and then programmer codes methods across various classes to implement use case

OOD steps:
1) choose single use case
2) Choose 1 of 3 paths: CRC cards, Communication diagram or Sequence diagram
3) package diagram construction: design classes into different subsystems

Design class diagram (DCD):
entity class: problem domain class, persistent class – exists after system shutdown
boundary (view class): designed to live on system automation boundary (UI, window classes)
controller: mediates b/w boundary/entity (between view and domain layer)
data access: retrieve from/send data to database



Symbols:
attributes, method signature, class level method and visibility

Method Signature: visibility, name, parameter list, return type

Attributes of classes should always be private except when you need to share class attribute with classes that inherit from your class. (Protected #)

Methods: usually public. Includes visibility, method name, parameters, return type

Class level method: underline class methods rather than object methods
Abstract class: cannot be instantiated. Only for inheritance (italics)
concrete class: can be instantiated

First cut:
Proceed **use case by use case**, adding to the diagram

- Pick the **domain classes** that are involved in the use case (see preconditions and post conditions for ideas)
- Add a **controller class** to be in charge of the **use case (1 for each)**
- Determine the **initial navigation visibility requirements** using the guidelines and add to diagram
- Elaborate the **attributes** of each class with **visibility** and **type**
- Note that often the **associations and multiplicity are generally removed from the design class diagram as in text to emphasize navigation, but they are often left on**

Object responsibility:
- objects responsible for carrying out system processing
objects know about other objects (associations), and about their attribute values

Separation of responsibilities:
- segregate classes into packages to focus on processing responsibility

Indirection:
- intermediate classes placed between 2 classes to decouple them but still link, controller class between UI and problem domain classes

Design principles:
Encapsulation: self contained unit that include attributes and methods
Obj Reuse: single purpose can be reused again – improving cohesion
Info hiding: data not visible
-Classes have dependence (coupling)

Cohesion:
measures considtency of functions in class (single focus, single focus of methods, method that performs multiple functions is not desirable)

High cohesion: greater focus on a single purpose
- classes should be highly ohesive

Effects of low cohesion: hard to maintain multiple function classes, hard to reuse, difficult to understand coz complex

Coupling:
degree to which different modules/classes depend on each other. How closely classes are linked
(parameters passed by methods, number of navigation arrows)- high coupling
Low/loose coupling easier to understand/maintain

classes should be independent and have low coupling

- **Summary**
    - Design class diagrams include additional notations.
    - Method signatures include visibility, method name, arguments, and return types.

- Decisions about design options are guided by some fundamental design principles like cohesion, coupling, indirection, and object responsibility.

Week 9 – Designing the User Interface

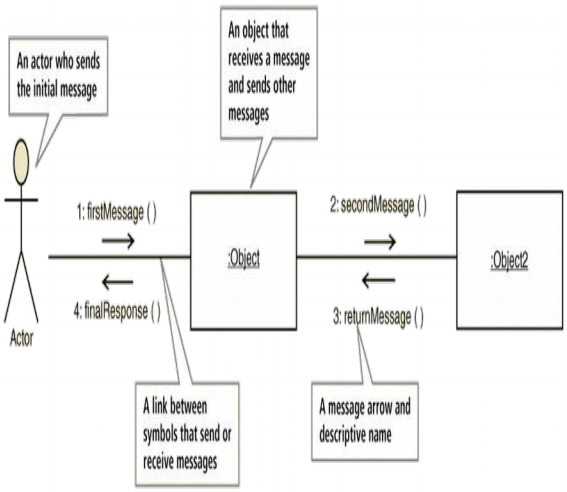
OOD steps:
1) choose single use case
2) Choose 1 of 3 paths: CRC cards, Communication diagram or Sequence diagram
3) package diagram construction: design classes into different subsystems

Use case controller:
- represents artificial class by designer
serves as collection point for incoming messages, intermediary between outside world/internal

Communication diagram:
actor: external role of person/thing initiating use case – sends messages
object: instantiated class obj that perform methods to execute use case, receive and process messages
link: connectors
messages: requests for service between 2 location (actor to destination)

- interaction diagrams: communication/sequence diagram
Focus on interactions between objects required in use case (messages)

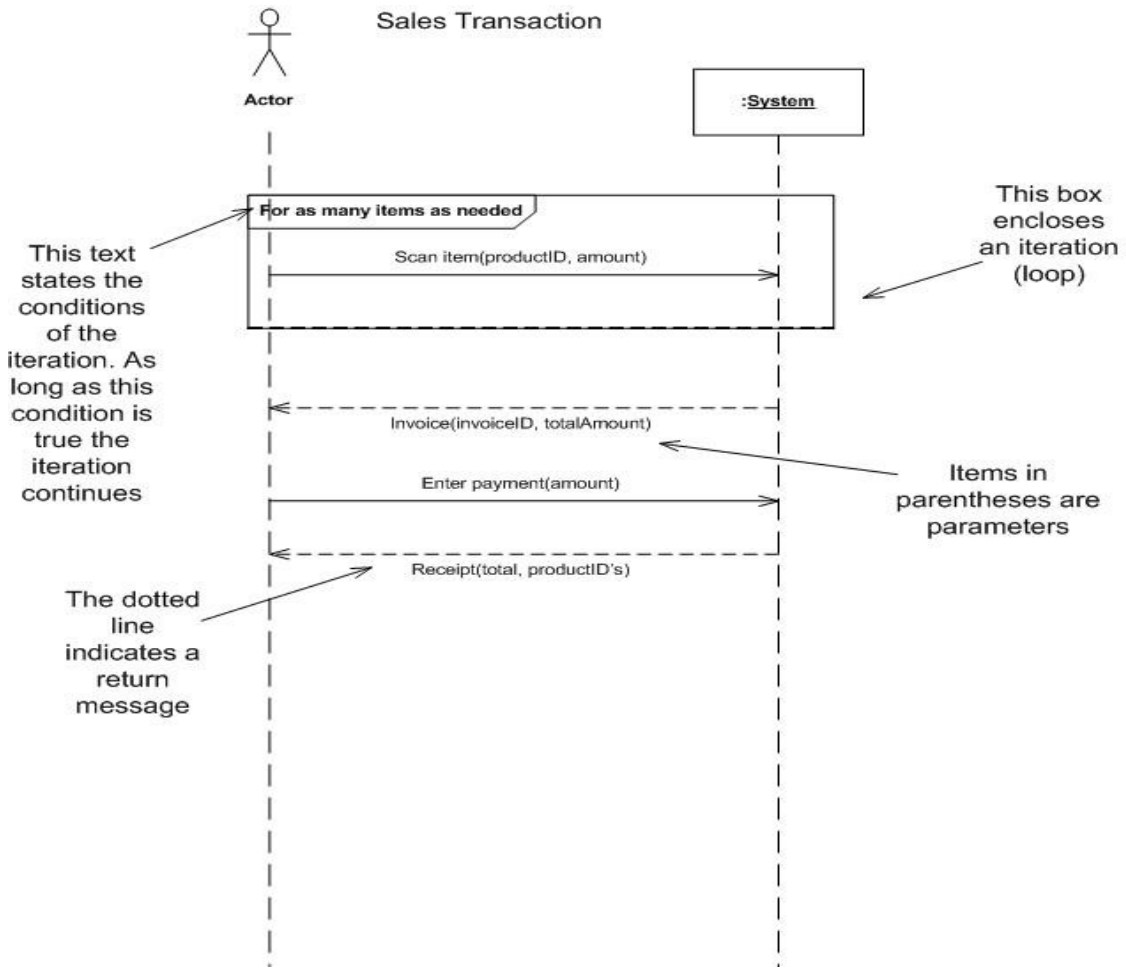
syntax:
T/F conditions: see if msg sent
sequence number
return value
message name
parameter list
multiply occurring message (*)



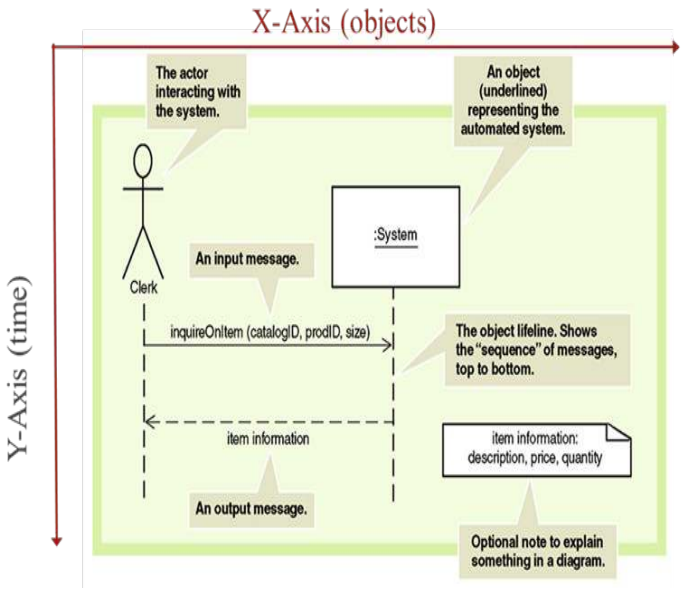
- Advantages
  - This type of diagram is useful when uses cases have **MODERATE COMPLEXITY**
  - That is when, use cases are not too large with many messages
  - It only provides a SNAPSHOT of CLASSES AND MESSAGES
- **Limitations**
  - Not much space it has for many messages
  - It can easily become cluttered with many messages

SSD – expressed at highest level, described in user terms and used in systems analysis
- sequence of interactions generated by actors from outside the system
- actor and one object in single use case
- time runs top to bottom
- one obj = complete system
- inner workings of system not immediately visible
- i/o messaging requirements for use case

components: actor, system, object lifeline & mess



Vertical line under object or actor:
-Shows **passage of time**
-The **Life-Line represents the object's life** during the interaction
-***No overlapping*** of vertical line associated with Actor and Systems object is permitted
-Long narrow rectangles: **Activation BOX** emphasise that object is **active only during part of scenario**
-*Return messages* are indicated using a dashed arrow with a label indicating the return value (requires :=)
-The return line is optional when nothing is returned



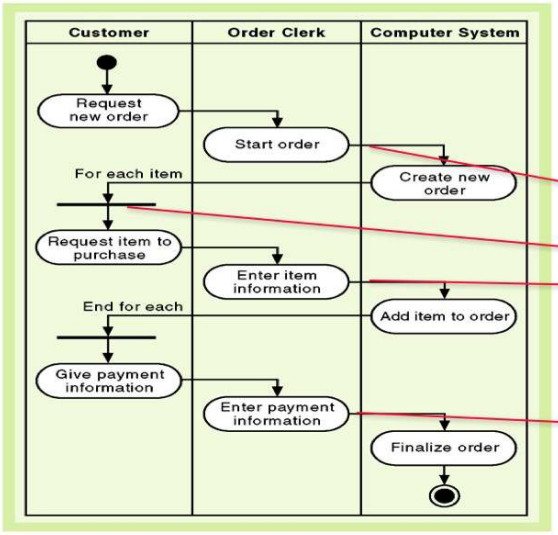
Example: Illustrates how objects interacts with each other.

Emphasizes time ordering of messages.

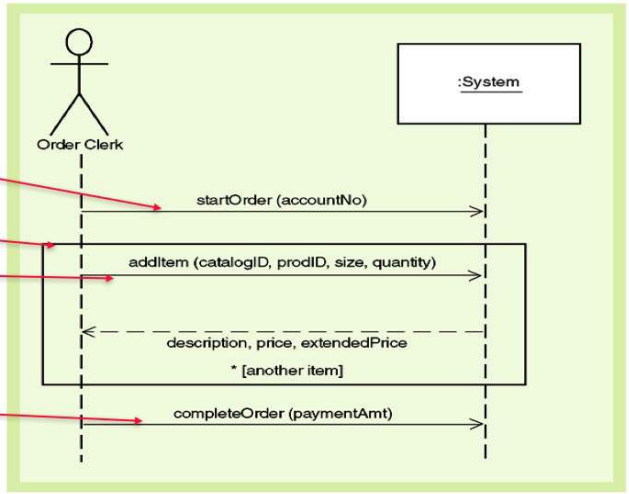
The Life-Line represents the object's life during the interaction
- **Time runs from top to bottom.**

Developing SSD:
- identify input msg using activity diagrams and use case descriptions
- describe messages from external actor to system (msg notation) – parameters/what system asked
- identify/add special conditions on input message (iteration/loop frame, opt/alt frame)
- identify/add output return messages (aVal:=getVal(valID), explicit return on separate dashed line

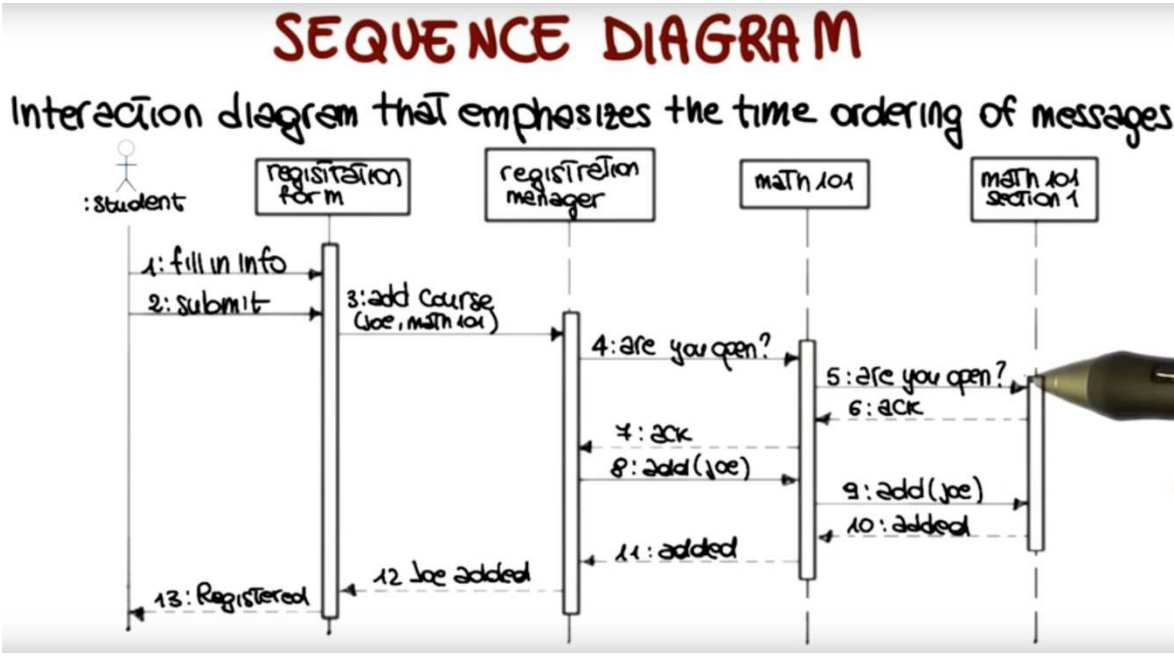
Example 1A: Simplified Activity Diagram of the Telephone Order Scenario



Example 1B: SSD of Simplified Telephone Order Scenario for Create New Order Use Case



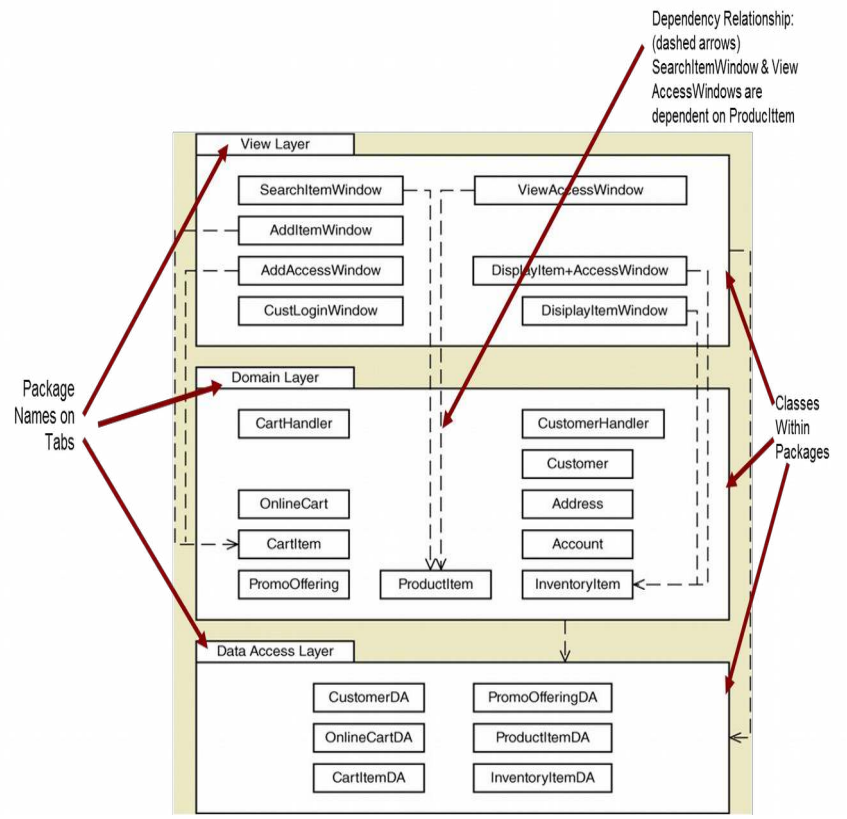
No return is shown



Sequence diagram	Communication diagram
Sequence diagrams are MORE DETAILED	Communication diagrams are less DETAILED Communication diagrams provide a BROAD "Overview" picture of the INTERACTIONS
Sequence diagram emphasises time in its layout	Communication diagram emphasises the objects that are communicating in its layout
Time is implicit in sequence diagrams (it is inferred by vertical position)	While, time is given explicitly in communication diagrams (via numbers).

Sequence diagram	System sequence diagram
The elements participating in a sequence diagram are objects (instances of various classes).	The elements participating (exchanging messages) in a system sequence diagram are Actors and Systems
It shows the details of what happens inside a system	SSD is actually a sub-type of sequence diagram made at the highest level
Sequence diagram is developed as a key activity of systems design	Elements of SSD should be described in user terms, without IT slang. Because it belongs to the requirements part of the project documentation

Package diagrams:
associates classes of related groups, shows dependencies between classes/packages
dependency relo: relo between packages/classes within package in which a change of the independent component may require a change in the dependent component (dashed line)
-determines final program partitions for each architectural layer (view, domain, data access) and can divide system into subsystem and show nesting



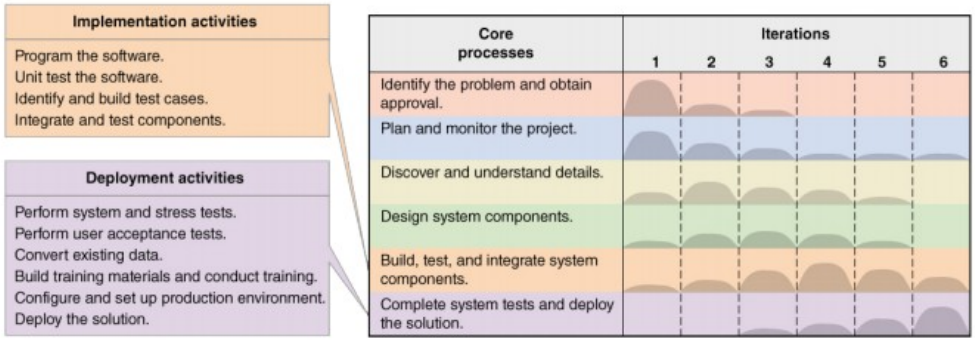
3 layer package diag:
dependencies- view layer depends on data access layer

- **Summary**
  - Three-layer design is generally used
  - Systems design is driven by use cases, and produces
    - design class diagrams
    - sequence diagrams
    - Communication
    - Package diagrams
    - Design class diagrams include artificial software classes (e.g. controller), in addition to domain class

Week 10 – Deploying the New System

Deploying the New System

Implementation and Deployment Activities



implementation: programming, testing

Deployment – system tests, stress tests, UAT/ converting data, training / setting up production env, deploying soln

Software bugs always arise due to human errors and misjudgement
conseq: expensive, dangerous and cause problems for company and people involved e.g. aircraft crash

Testing Concepts

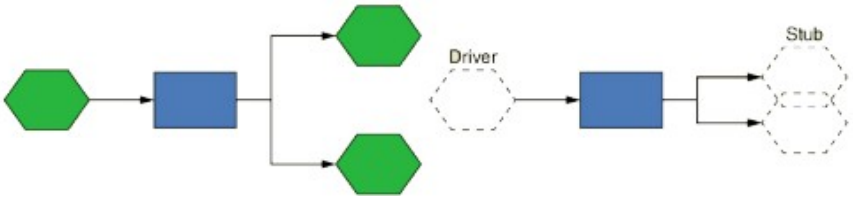
- Testing
  - The process of examining a component, subsystem, or system to determine its operational characteristics and whether it contains any defect, uncover design flaws/limitations and reduce risk of failure (and to determine if it is fit for purpose)
- Test Case
  - Formal description of a starting state, one or more events to which the software must respon or ending state
    - Defined based on well understood functional and non-functional requirements
    - Must test all normal and exception situation
- Test Data
  - A set of starting state and events used to test a module, group of modules, or entire system
    - Data will be used for a test case

Test type	Core process	Need and purpose
Unit testing	Implementation	Software components must perform to the defined requirements and specifications when tested in isolation—for example, a component that incorrectly calculates sales tax amounts in different locations is unacceptable.
Integration testing	Implementation	Software components that perform correctly in isolation must also perform correctly when executed in combination with other components. They must communicate correctly with other components in the system. For example a sales tax component that calculates incorrectly when receiving money amounts in foreign currencies is unacceptable .
System and stress testing	Deployment	A system or subsystem must meet both functional and non-functional requirements. For example an item lookup function in a Sales subsystems retrieves data within 2 seconds when running in isolation, but requires 30 seconds when running within the complete system with a live database.
User acceptance testing	Deployment	Software must not only operate correctly, but must also satisfy the business need and meet all user "ease of use" and "completeness" requirements—for example, a commission system that fails to handle special promotions or a data-entry function with a poorly designed sequence of forms is unacceptable.

Unit Testing

- Test of an individual method, class or component before it is integrated with other software
- Driver
  - A method or class developed for unit testing that simulates the behaviour of a method that sends a message to the method being tested
- Stub
  - Method or class developed for unit testing that simulates behaviour of a method invoked that hasn't yet been written.

- Driver and stub components



Integration Testing

- Test behaviour of a group of methods, classes or components
  - Interface incompatibility - E,G - one method passes a parameter of the wrong data type to another method
  - Parameter values - A method is passed or returns a value that was unexpected, such as a negative number for a price
  - Run-time expectations - A method generates an error, such as 'out of memory' or 'file ready in use', due to conflicting resource needs
  - Unexpected state interactions - the state of two or more objects interact to cause complex failures, as when an OnlineCart method operates correctly for all possible Customer object states except one

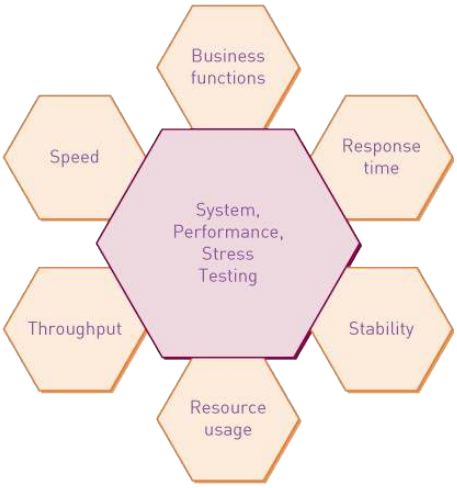
- Integration testing of OO software is very complex because an OO program consists of a set of interacting objects including methods/classes
  - Methods can be called by many other methods, and the calling methods may be distributed across many classes
  - Classes may inherit methods to be called is dynamically determined at run time based on the number and type of message parameters
  - Objects can retain internal variables values between call. The response to two identical calls may be different due to state changes that result from the first call or occurs between calls

- Required Procedures
  - Build and unit test the components to be integrated
  - Create test data - detailed test data, must be coordinated between developers
  - Conduct the integration test - Assign resource and responsibilities. Plan frequency and procedures
  - Evaluate the test results - identify valid and invalid reponses
  - Log the test results - log valid test runs
  - Correct the code and retest

System, performance and stress testing

- System testing
  - represents an *integration test of an entire system or independent subsystem*
- System tests can be performed
  - at the end of each iteration
  - more frequently
- The system is completely compiled and linked (built), and a battery of tests is executed to see *whether anything malfunctions in an obvious way ("smokes")*
- Automated testing tools are used.

- Catches any problems that may have come up since the last system test
- **Performance test or stress test**
  an integration and usability test that determines whether a system or subsystem can *meet time-based performance criteria*
- Response time
  the desired or maximum allowable time limit for software response to a query or update
- Throughput
  the *desired or minimum number of queries and transactions that must be processed per minute* or hour



User Acceptance Testing (UAT)

- A system test performed to determine whether the system fulfils user requirements, may be performed near ending of project
- Very formal activity in most development projects
- Details of acceptance tests are sometimes included in the RFP and procurement contact
- Plan the UAT
  - Should be done early in project
  - Test cases for every use case and user story
  - Identify condition to verify that the system supports the use case accurately and completely
  - Document/track results
- Prep and Pre-UAT activities
  - Develop test data
  - Plan and schedule specific tests
  - Set up test environment
- Manage and execute the UAT
  - Assign responsibilities
  - Document and track results
  - Rework the plan for re-testing

- Log and Results tracking list

Spec ID	Cross refer to use case	Short description	Test condition	Expected outcomes	Name of tester	Date executed	Acceptance criteria	Status	Outstanding issues
10	101	Maintain customer info	Add customer, update customer, delete not allowed	New customer with all fields, updated customer with selected fields	Mary Helper	7/15/2015	All expected outcomes, DB updated successfully	Accepted	None
11	201	Maintain sale info	Create sale, upate sale, finalize sale, pay for sale	New sale in DB, update selected fields, payment creates transaction	Mary Helper	7/15/2015	All expected outcomes, DB updated successfully	Pending	1005, 1006
12	202	Ship items	Display items, update status	Sale update, sale items updated, shipment created				Not started	

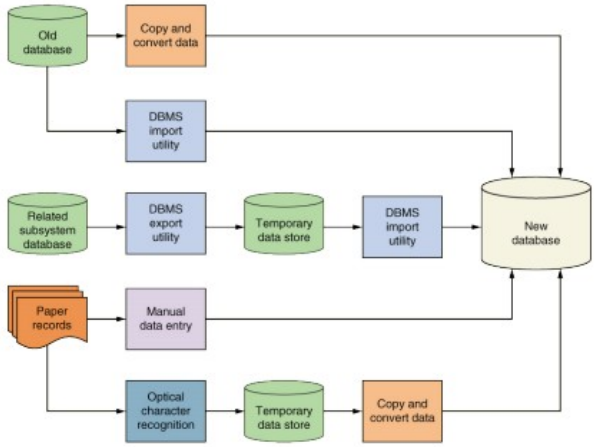
Benefits of testing:
1) better quality of software
2) more value to customers/users
3) satisfied and happier customer
4) improved customer experience: easy to understand and use

Converting and Initializing Data

OS requires fully populated database to support ongoing processing

- Data needed at system start up can be obtained from these sources:
  - Files or database of a system being replaced
  - Manual records
  - Files or databases from other system in the org
  - User feedback during normal system operation
- Reuse existing databases
  - Modify or update existing data
- Reload database
  - Copy and convert the data
  - Export and import data from distinct DBMS
  - Data entry from paper document

- Complex data conversion example



Step 1- convert data into new format
2- identify relevant data from various system
3- identify/capture data from paper/manual sources

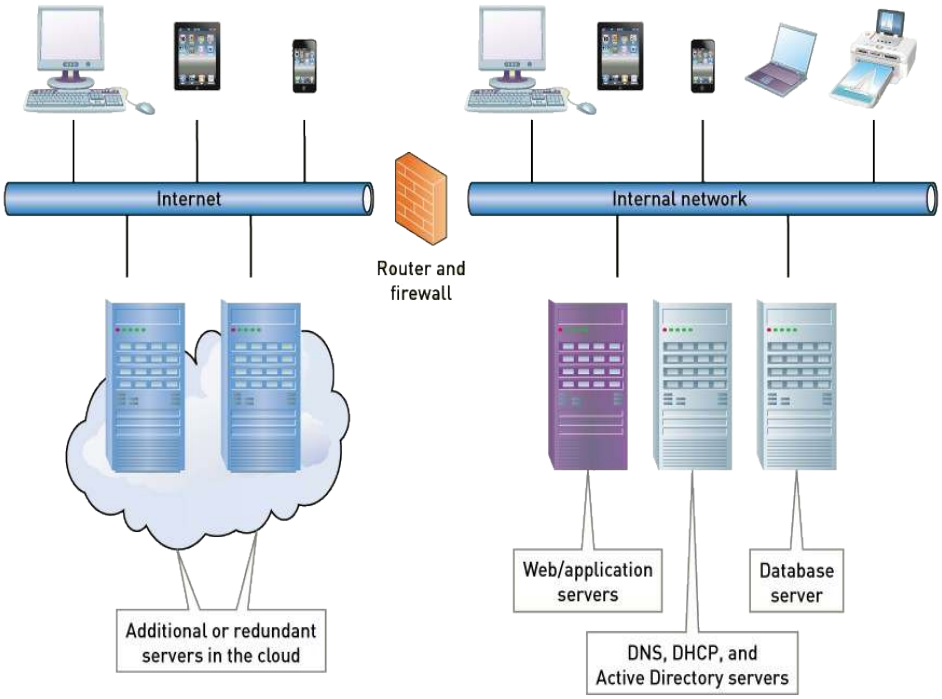
Training Users

- Training for end users must emphasize hands on use for specific business processes or functions, such as order entry
  - Widely varying skill and experience level call for at least of some hands on training
- System operators training can be less formal when the operators aren't the end users

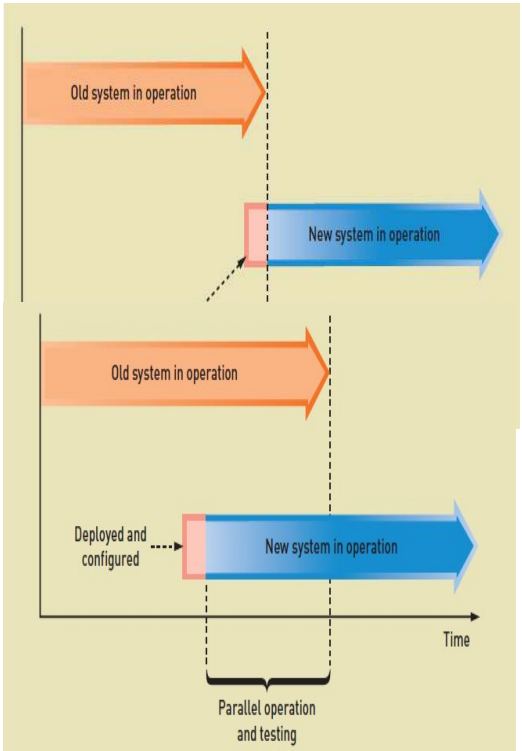
Training Users

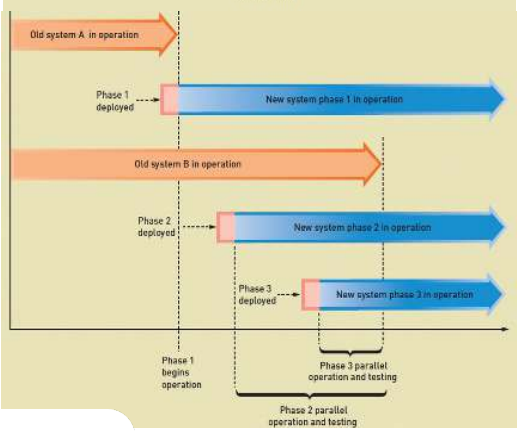
End-user activities	System operator activities
Creating records or transactions	Starting or stopping the system
Modifying database contents	Querying system status
Generating reports	Backing up data to archive
Querying database	Recovering data from archive
Importing or exporting data	Installing or upgrading software

- System Documentation
  - Description of system requirement and architecture to help maintenance and update of the system
- User Documentation
  - How to interact with and use the system for end user and system operator
- System Document
- User Documentation: online files and videos maintained/developed



- This figure shows a typical infrastructure needed for a web-based application
- Applications can be written using an OO programming language like Visual Basic
- Parts of the application are stored in application servers
- A database server is needed to manage a database that is accessed by the application
- Active Directory server is needed to verify:
  - Users' access to the application
  - Domain naming system





- Deployment approaches: direct, parallel and phased
  Direct: new system installed, made operational and immediately turns off any overlapping system (high risk, low cost)

  Parallel: operated old and new for extended time period (lower risk, high cost)
  - incompatible inputs, heavier load on equipment/staff
  (Partial parallel – process subset of data, use only parts of system)

  Phased: installs new system, makes operational in series of steps/phases

  Versioning

  Test version: common to have multiple version of systems
  system version created during development called test version

  test version evolves
  alpha version: test version thats incomplete, ready for some level of rigorous integration/usability testing (lifetime short)

  beta version: stable, tested by end users, over extended time, done at real time/real data and more complete than alpha

- product version, release version/product release:
  system version that's formally distributed to users, made operational for long term use, it's the final product

  - track of version needed, must be identified and archived,

Summary

- Implementation and deployment are complex processes because they consist of so many interdependent activities
- Testing is a key activity of implementation and deployment
- Versioning improves post deployment support by enabling developers to track problem support to specific system versions
- Three options for deployment: direct deployment, parallel deployment, and phased deployment
- Direct deployment is riskier but less expensive. Parallel deployment is less risky but more expensive
- For moderate to large projects, a phase deployment approach is preferred to get key parts of the system operational earlier

Week 11 – Securing/maintain (post-implementation) the software system

Software security ensures: risks reduced and improved software reliability
risk types:

Risk 1: Human error • wrong data entry • accidental data deletion	**Risk 4:** Fraud • deliberate attempts to *corrupt or amend legitimate* data
Risk 2: Technical errors • h/w failure • s/w crash	**Risk 5:** Commercial espionage • Unauthorised access to sensitive data (e.g. price, credit card details)
Risk 3: Accidents and disasters • Flood • Earthquake • Fire	**Risk 6**: Malicious damage by internal employees, external hackers • Change software settings • Change user interface

Information security: Confidentiality Integrity Availability (CIA)
C- authorised users view data, data encryption
I- prevent data modification by unauthorised personnel
A- accurate, authorised and timely access to data

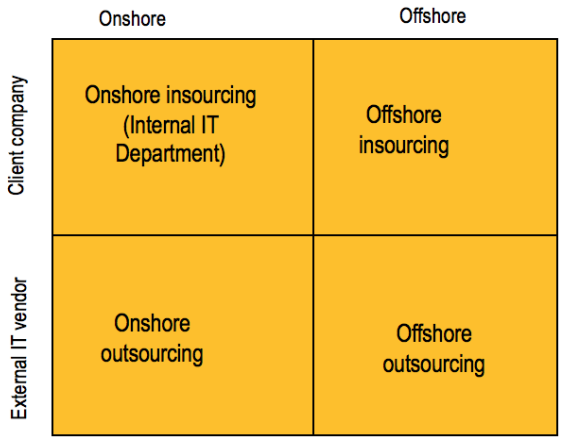
Security:
1- prevention, security controls
2- detection
3- deterrence: discourage security breaches, training
4- data recovery

Technical controls: software to authenticate/authorise
policy controls: procedural controls (staff training), regulatory controls (legislation), integrity controls (input controls)

Input controls: invalid/erroneous data prevented to enter
- value limit controls, completeness controls, data validation and field combination controls

Output controls: arrives at proper destination, accurate, current and complete

SOURCING: process of finding suppliers for materials
IT context –process involved deciding sources (internal IT dept, external vendors or mixed) that would satisfy IT req of organisation
- process of subcontracting IT function of org to third party
- or
practice of transferring IT assets to third party vendors for delivering services of IT functions to third party vendors
- or
passing ownership/control (partial/full) of IT functions previously performed in house to outside contractors, hands over mgt responsibility



Insourcing: IT function/systems maintained and developed in house
outsourcing: subcontracting IT function (part/full) to third party local or overseas

(both can occur onshore or offshore)

Transfer of IT function includes: asset (hw,sw) control (decision rights), staff – governed by legal contract
contract – short term, long term
must describe services vendor provides, how delivery is measured, penalties for non performance, financial arrangements, and legal issues with IP/privacy

Benefits: reduced costs, strategic (client focus more on core activities), access to latest skills/tech, increase in share price/market reaction (savings passed to shareholders hence increases share price)
disadvantages: loss of distinctive competencies (can build similar for rival), privacy concerns (air-tight contract needs to be formulated), increased vendor dependency (monopoly power to demand high price in future), job loss/impact on economy

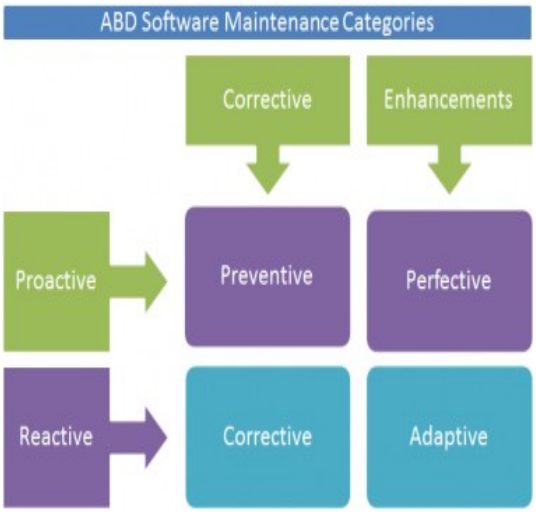
Build in house:
develop, maintain and meet req of business
adv: customisation, train IT staff with new skills
dis: time taken for developing, testing, upgrades, training and cost

- commercially available, standardised and bought from software vendors, can customise slightly (value added reseller - VAR), can be used across multiple orgs, vertical application (developed to meet a particular type of business), can have both

Reasons to buy:
- needed for standard business function e.g. payroll
- resources short in supply
- more cost effective
- needed in short time
- maintained by vendor, no internal development is necessary
- upgrades done by vendor
- high quality
- political: external work perceived superior than in house, cuts source of internal conflict

Limitations to buy:
- can’t meet all requirements
- need customisations, vendors can refuse to support customised packages after, and can be so extensive its cheaper to develop in-house
- conversion/integration costs so high, can be infeasible

Package selection steps:
1- identify products that match requirements
2- solicit, evaluate and rank vendor proposals
3- select best vendor
4- integrate vendor product with existing system



Post implementation:
- maintenance, post implementation review
mountainous costs are high, greater than implementation costs

software maintenance is last step of SDLC: criteria- degree of activeness, nature of maintenance

maintenance types: corrective, adaptive, perfective and preventative

corrective: rises after deployment, initiated by bug reports, focus on removing defects rather than adding new, corrective maintenance declines over time as bugs found

adaptive: change in software that takes place to make software respond to new environment
environment – conditions/influences which act (inside/outside) on the system

perfective: meet user req not previously recognised or given high priority (missed in dev, not known about or considered unimportant).

Preventative: identify/fix problems notes while other errors
defects found and corrected before damage caused, reduce change of future system failure

PIR: analyses what went right/wrong with IT project
conducted 2-6 months after deployment
issue 1: look at orig req and evaluate how they were met
2: compare cost of dev and op cost against original estimates (maintenance costs?)
3: compare original/actual benefits
4: new system reviews to see whether more of original or additional benefits can be realised

- **Summary**
  - System and information security
  - Confidentiality-Integrity-Availability (CIA) security model
  - Various types of sourcing
  - Types of maintenance
  - Post-implementation review

Week 12 – review

- Example Question:
  - Why is user interface important?
- Correct Solution:
  - Discuss the reasons of user acceptance, usability, and possibility of systems failure