Week1: Introduction

Conventional software development is using **waterfall model:**

- Requirements analysis
- Design
- Implementation
- Testing

Main problems with this model:

- It is difficult for customers to identify detailed criteria at the beginning of the project. **Revision of requirements is difficult.**
- product usually is not available for customer's feedback until the end of the development, thus **high risk**

Agile approach: (main features**)**

- Uses **incremental and iterative** approach to software design
- Allows **self-organizing teams** to respond to change
- Enables the customer to have **early and frequent opportunities to give feedback** on the product
- Implements small projects very quickly
- Enables errors to be fixed in the middle of the project.
- Gives less priority on documentation rather software development
- Delivers shippable features to the customer at the end of an **iteration**(**sprint**)

Agile software development – covers a set of methods:

- Scrum
- Kanban
- Extreme programming (XP)
- Dynamic Systems Development Method (DSDM)
- Feature driven development (FDD)
- Adaptive Software development (ASD)

Scrum method: Three roles:

- Product Owner –
  - knows what the customer wants and can then translate the customer's needs back to the Scrum team
  - must have the authority to make all decisions necessary to complete the project
  - is responsible for identifying product features, translating them into a prioritised list (i.e. Product Backlog), prioritising the list for the next Sprint (i.e. Sprint Backlog)
  - working with customers and stakeholders as well as working with the Team implementing the Product Backlog
- Scrum Master responsible: (a person who works closely with the team)
  - to guide the team
  - to ensure the team understands the goal
  - to facilitate the Scrum ceremonies(sprint planning meetings)
  - to coach the team
  - to make sure the team is cross-functional (full-stack of specialist, responsible for different project parts)
  - to assist the Product Owner improving the backlog
  - to keep team in high morale
- The Team

Scrum method contains 4 main meetings:

1. Sprint planning (ceremony or iteration) – creating sprint backlog
2. Daily scrum 15-minute meeting (with 3 main questions)
   a. What have you done?
   b. What are the obstacles?
   c. What will you accomplish today?
3. Sprint review
4. Sprint retrospective

Sprint backlog – features to be done in this sprint

Product backlog – list of **all** features required **by customer**
Sprint duration is always the same and repetitive! New functionalities are shown after every sprint.

Agile: **Kanban method** mainly know for KANBAN board that contains the process steps to deliver work. Applies Limiting work-in-progress (WIP), which means you are able to have only 1 task working on.
Extreme programming (XP) is done pair-wise when one is active on generation of code and another one supporting and reviewing it. It is test based approach to requirements and quality.

Test Driven Development (TDD)
TDD cycle:

- Write the test for a new feature (function or requirement through user stories)
- Run the test and see if the new test fails
- Write the code that pass the test that fails
- Refactor code: improve code
- Repeat the above process

Same: TDD and agile both adopt an interactive and incremental process
Difference between **TDD** and **Agile:**

a) TDD focuses on how code is written and tested, whereas agile focuses on the overall development process
b) TDD focuses on how code is written and tested by a given developer, whereas agile focuses on groups of developers

User stories: (It describes the type of user, what they want and why**.)**

- A description by user of a software feature (**template**: as a <usertype> I want to <some task> so that <achieved goal>)

Examples:

- ✓ as a student, I want to see my fee, so that I know the remaining balance
- ✓ as a user I want the ability to restore my password.
- ✓ as an administrator I want to be able to create a new user to the team when needed.

Acceptance criteria used by the software team to simplify the understanding of the client's intent
In other words – **Acceptance criteria:** (user story writen in precise steps/details)

- Balance displayed
- Balance calculated and etc.

Epics are large user stories (which usually are splitted into smaller ones)
Themes are a collection of related user stories!

Another example:
User story - As an online shopping site customer, I want to be able to register online, so I can get regular updates
Acceptance Criteria –

- ✓ I can enter my name and contact details to register with the site
- ✓ I can choose what services (news, discounts) to receive
- ✓ I will receive an acknowledgment for the registration

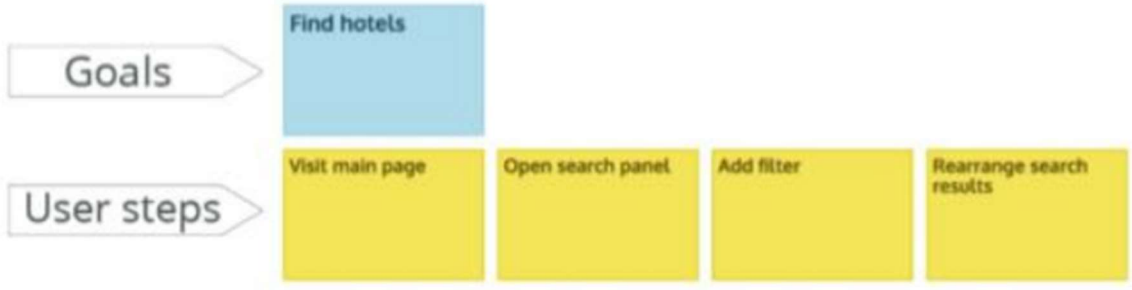
Story points are needed to measure the amount of effort required to make that story! Usually are assesed using T-shirt sizes or numeric estimation

User story mapping

A technique to predict the entire software product as a series of steps in which the user completes tasks. It enables user stories with solutions to be grouped

Example:

1. identify the goals
2. mapping the steps to achieve the goal (in other words you split goal in a steps)

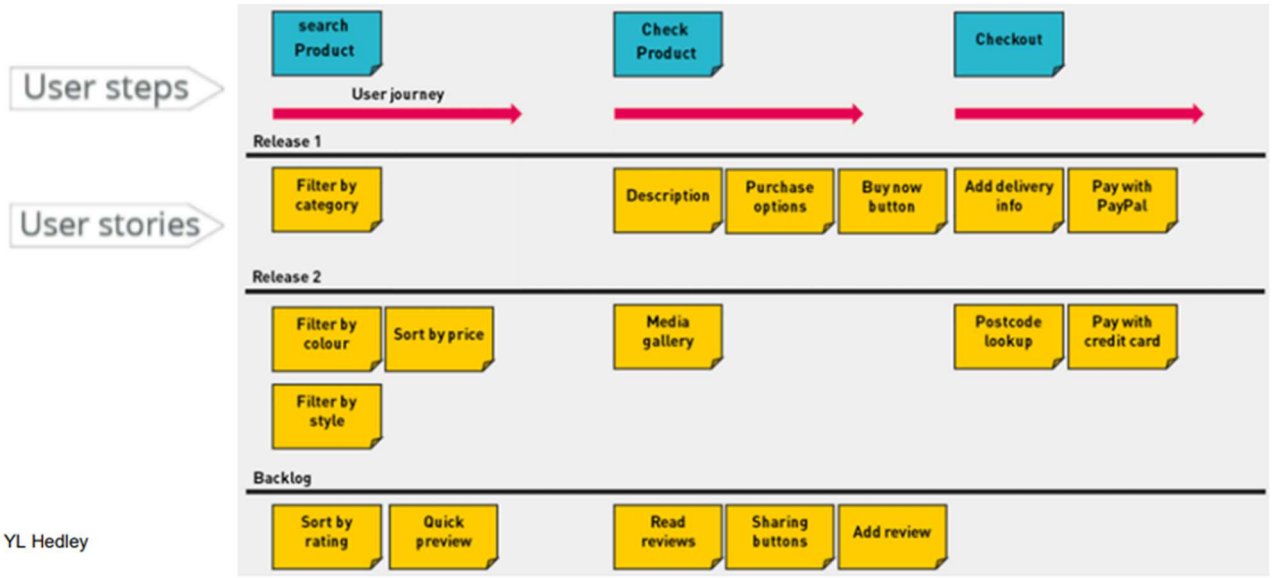


3. Then you split each step even more(into user stories): i.e. open search panel:
    a. Search by location
    b. Search by name

Then we gather user stories to make releases.

Goal: Purchase a product – to further break down the user stories/solutions, re-prioritise them into releases

Those are arranged usually by story points. Story points measure the effort/**impact** to product (size and complexity) of a user story relative to others to implement it, not time needed to complete it



Story points can be expressed in:

- numeric values: Fibonacci sequence
- non-numeric values, e.g. small, medium, large

Estimation Techniques:

- Planning Poker: to estimate the work to be done
- T-Shirt Sizes
- Relative Mass Valuation

The above three allow a team with no prior knowledge of point values to reach a understanding of the relative value of all the different stories.

Planning Poker:

- Played by team members during planning meetings
- Each team member given a set of cards with numbers on them, ordered from 0 to 21 using the **Fibonacci sequence.** In practice, the numbers 0, 1, 2, 3, 5, 8, 13, 21, 40, 100 are used.

A story estimated as a 2 should be about one fourth as difficult as a story estimated as an 8. Stories estimated at 20 or higher may be so large that they need to broken up into smaller stories before they can be attempted. Stories estimated at 0 may not even be worth tracking.

T-Shirt Sizes:

- Benefits: can be very effective for teams just starting out with agile
- Limitations:
  - Lack of a mathematical relationship between different measurements, such as a medium and an extra-small.
  - May still need to be converted to numerical values at a later stage for tracking effort over time and charting an estimated velocity for the team.

Relative Mass Valuation:

- user stories are estimated relative to each other in size, not on the basis of hourly effort
- quick way to go through a large backlog of user stories and estimate them all based on how they relate to each other.
- to reach a rough point estimate, not a precise order

Velocity 1

- ✓ a measure of a team's capacity to get work done in a given iteration (or sprint);
- ✓ expressed as a range of numbers, e.g. 23 to 32 story points per sprint, especially early on in a project's life
- ✓ used to plan releases and adapt work as progressing through a project, so as to adjust the forecast for completion regularly and accurately through execution

Velocity 2

- ✓ breaks down user stories for a sprint into tasks. Estimate the number of hours each task will take, which includes design, development, testing, etc.
- ✓ assesses how much capacity the team would have in a given sprint

Velocity 3

1. Using task hours:
   Assume using a capacity of 70 percent for a team as a baseline
   - if the total hours available to the team = **4 team members * 2 weeks * 40 hrs per week = 320 hours**
     - Multiplied by 70 percent capacity = 224 hours
2. Using story points:
   Assume that 36 story points for the completed features and a capacity of 20 percent for a particular sprint
   - apply 20 percent (36*20% = 7) either side to get a range of the lowest and highest, (i.e., 36-7 and 36+7)
     an estimated velocity = 29 to 43 story points

Task Breakdown:

- To **use hours to estimate the amount of effort required to complete a Task**; whereas Story Points to estimate the amount of effort required to complete a User Story
- To consider **the architecture and its components** required by the system when breaking down a user story to tasks

Example: Architecture Components

User story: As a user, I want to be able to search a product, so that I can find the product I am looking for

Based on the architecture (including components), the tasks may have:

a) Add a new Product Search page
b) Add a function to sort the search results of a product in order according to criteria
c) Add a function to filter the criteria for searching for a product

Task Breakdown:

- **To create meaningful tasks**. For example, instead of having 'coding' as a task, the tasks can be given as 'Develop the login class' or 'Create a user table and save the login data to the database'
- **To create tasks that are right-sized**. As a general guideline for the size of the tasks is to have tasks that require less than 8 hours. Avoid to break down the tasks that are very small, to a minute level such as 10, or 30 minutes etc.

Task Breakdown:

- **To use the Definition of Done (DOD)** as a checklist to produce a list of tasks. The DOD defines the completeness criteria of a story. It includes all items that have to be completed for a story ready to be delivered to the customer or user.
- For example, DOD may include 'Coding tasks completed' or 'Unit testing written and passed'. The DOD can then be used as a checklist to come up with tasks for the user story and allows the agile team to take on each one of them and complete.

Example: DOD

User story: As a user, I want to be able to search a product, so that I can find the product I am looking for If the Definition of Done is identified as below:

a) Design for product search is completed
b) Code for product search is completed
c) Automated UAT (User Acceptance Tests) for product search is passed
d) Documentation for product search is updated

Based on the above DOD, the tasks may include:

- Produce the design model for product search function
- Write code for the product search page
- Write and run UATs for the product filtering feature
- Document the product search features

Kanban

Kanban board is created to monitor current status of development. It **contains these columns:**

a) To Do: lists the tasks that are not yet started. (i.e., backlog)
b) Doing: the tasks that are in progress
c) * Testing. Tasks that are done but not yet confirmed
d) Done: the tasks that are completed

Scrum: SprintPlanning and Tracking

Burn-downs charts: shows how much work is remaining to be done. It has multiple **approaches**:

- **using effort remainin**g approach: A task breakdown during the sprint planning meeting reflects progress assuming that all tasks will be completed within the sprint at a uniform rate

Each task **should have associated hours** to complete and to plot the ideal burn-down chart based on the task breakdown

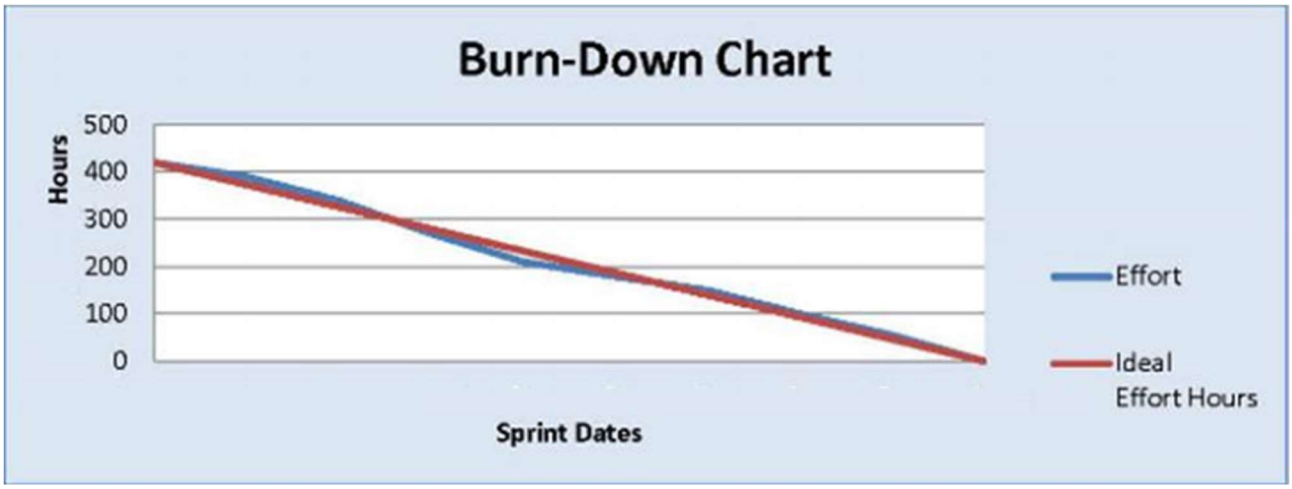
Each member carries out tasks from the task breakdown and update effort remaining for the tasks.

Example: if the total estimated effort for Task 1 is 8 hours at the start of a sprint, after working on the task, if the team member believes that it requires another six hours to complete, the "Effort Remaining" should be updated as 6.

Story Name	Task No	Description	Status	Owner	Estimated Effort (in Hours)	Estimated Remaining (in Hours)
Story 1	1	Create a Product Search page	In progress	Developer 1	8	6
	2	...	...	...	...	...

- **using story points**

Ideal chart should look like this:



Methodology

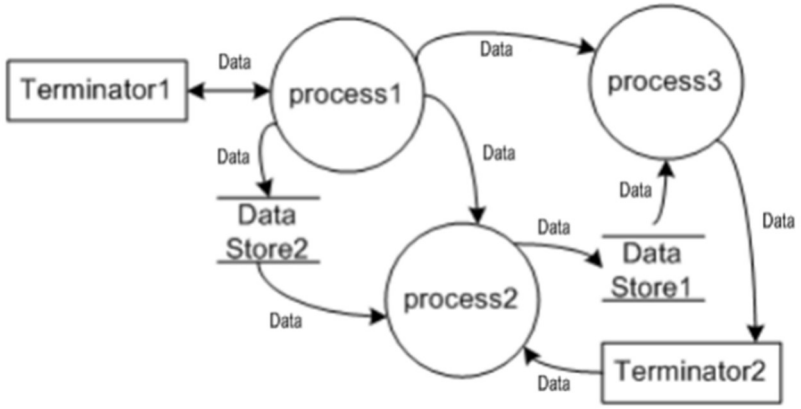
commercial **Computer Aided Software Engineering (CASE)** tools often used (such as Visual Paradigm for business process modelling and object oriented design)

Software design and analysis can be:

- Structured (function-oriented): 1970s
- Object-oriented:1990s

Structured Methodology

- Considers the **processes and data separately**
- **Processes** manipulate **data** and show how processes transform data that flow through the system



Modelling Techniques: (1. Data Flow Diagram)

- Data Flow Diagram (DFD): models how data moves around a system, particularly used in modelling a business process flow
- DFD examines:
  - processes (activities that transform data from one form to another)
  - data flows (routes by which data can flow)
  - data stores
  - external entities

1. Applications in agile development: (DFD)

- ✓ can be used to visualize and understand **business and technical requirements** and plan the next steps.
- ✓ can be a simple yet powerful tool for **communication and collaboration** for rapid software development.

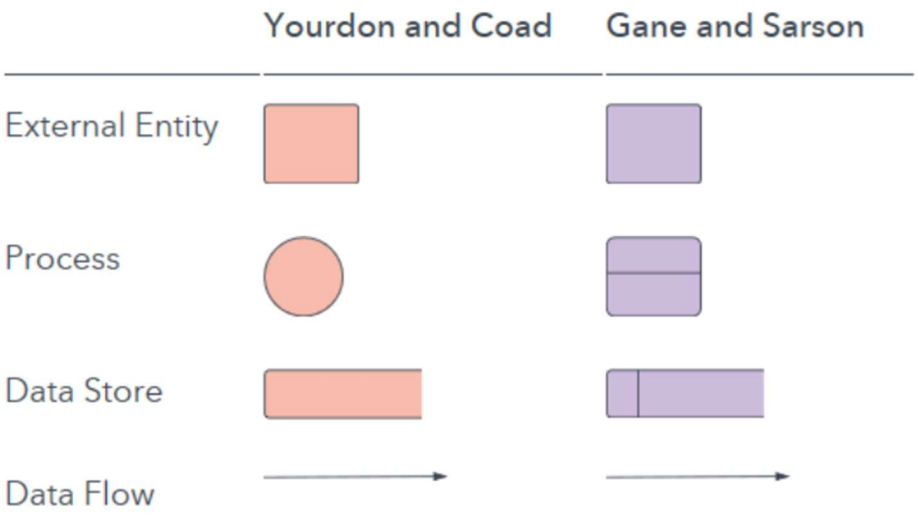
2. Applications in software engineering:

- ✓ System analysis: **logical DFDs are used to describe the current state and structure of a system** whereas physical DFDs are to show the solution of software, devices and other system interaction.
- ✓ Software design: **logical DFDs are used to model a set of activities and functions**, whilst physical DFDs model the implementation of a system that fulfils the requirements.

3. Applications in business analysis and process modelling:

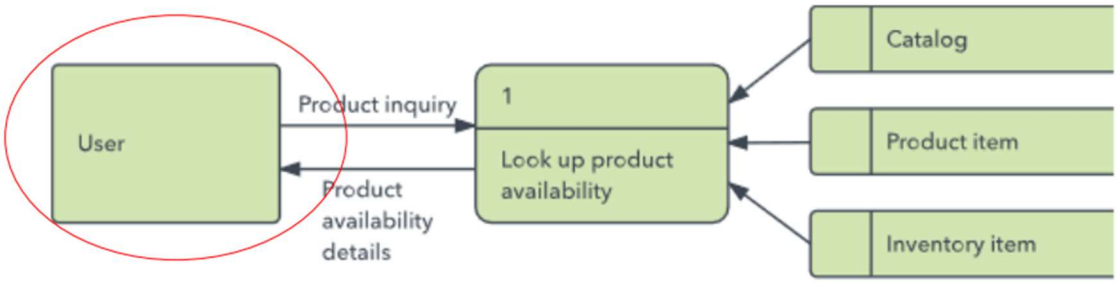
- ✓ **logical DFDs** are used to help capture **business** requirements and serves as a clear communication tool with clients in the business activities,
- ✓ **physical DFDs** are used to demonstrate how the system would **be implemented to meet the requirements.**

Data flow diagram notations:



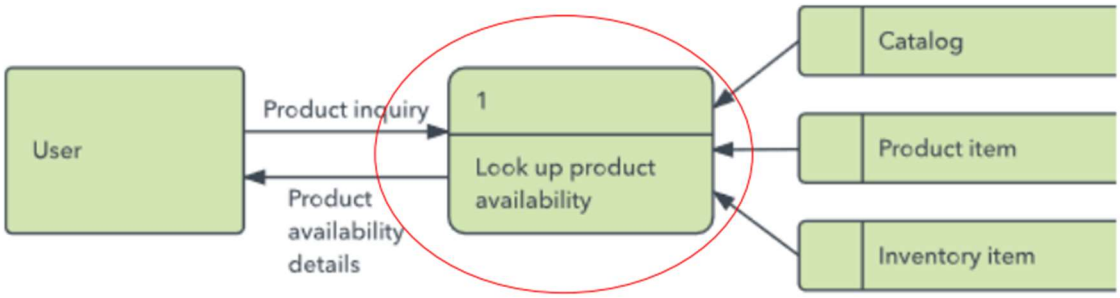
External Entity: Gane-Sarson notation

- ✓ is also known as actors, sources or sinks, and terminators
- ✓ produces and consumes data that flows between the entity and the system, with data flows being inputs and outputs of DFD
- ✓ are external to the system and can also represent another system or a subsystem



Process: Gane-Sarson notation

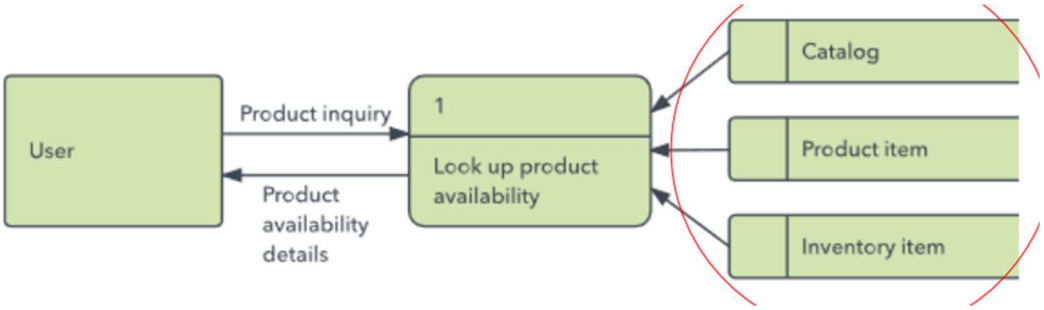
- ✓ an activity that changes or transforms data flows
- ✓ has inputs and outputs
- ✓ typically oriented from top to bottom and left to right on the diagram



Data Store: Gane-Sarson notation

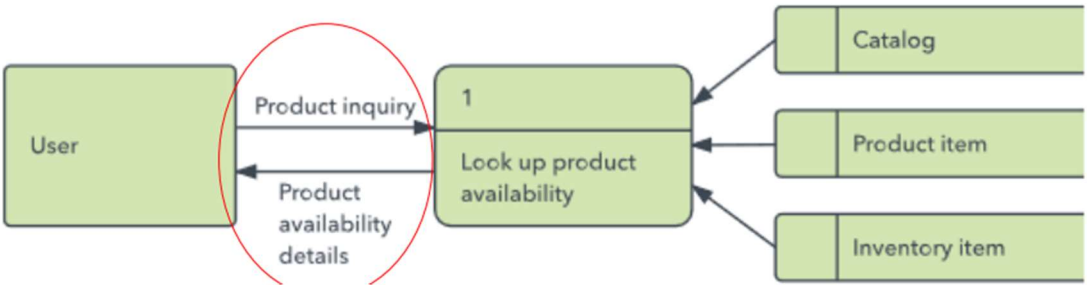
- ✓ holds data for access or to be processed.
- ✓ input flows to a data store include information or operations that change the stored data

- ✓ output flows would be data retrieved from the store



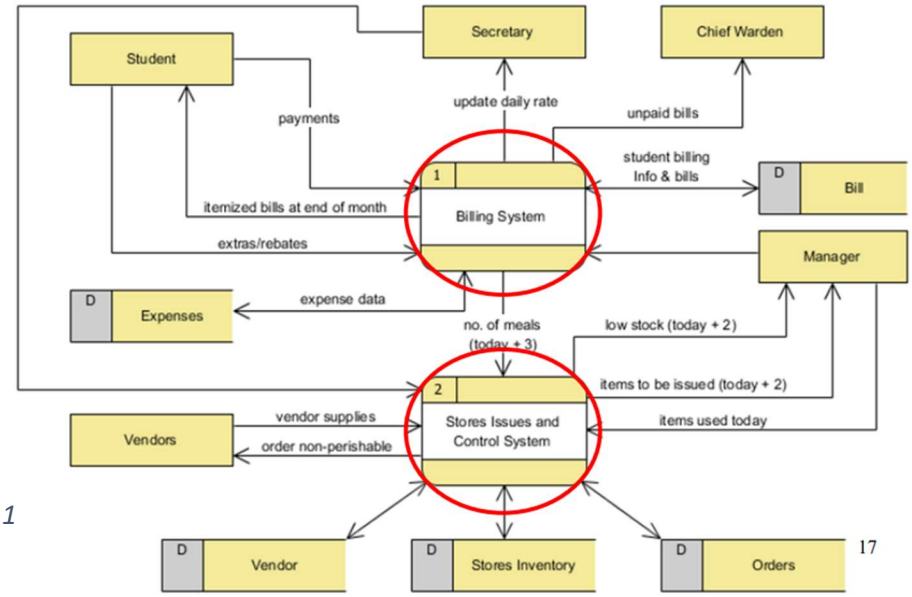
Data Flow

- ✓ indicates the direction of flow of data between external entities, processes and data stores
- ✓ input and output data flows are labeled based on the type of data or its associated process or data store



DFD Level 1

- ✓ provides a more detailed breakdown of processes in the Level 0 diagram.
- ✓ highlights the **main functions** carried out by the system and breaks down the high-level process into a set of **sub-processes.**

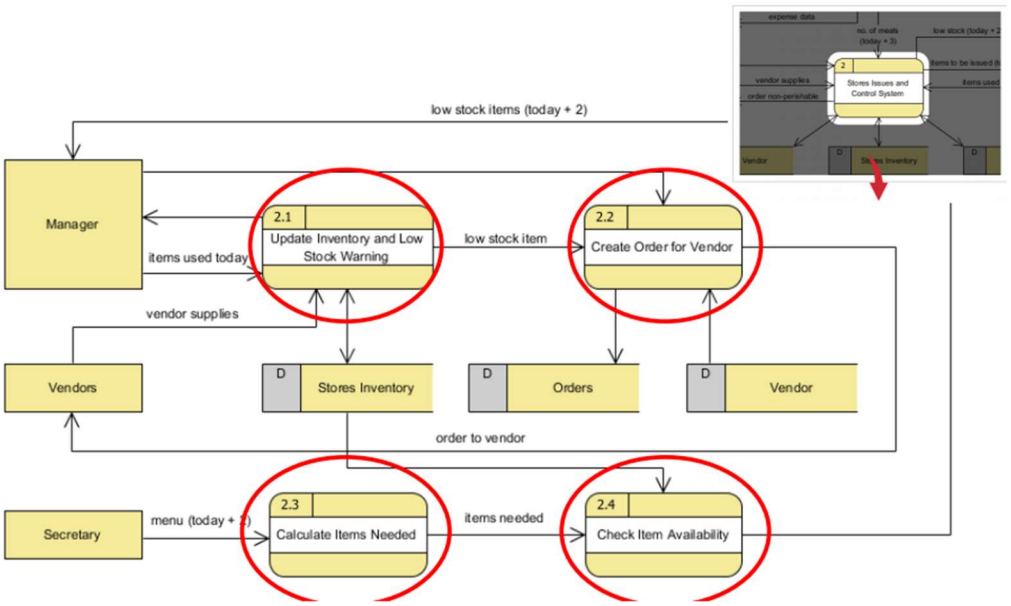


Example 1

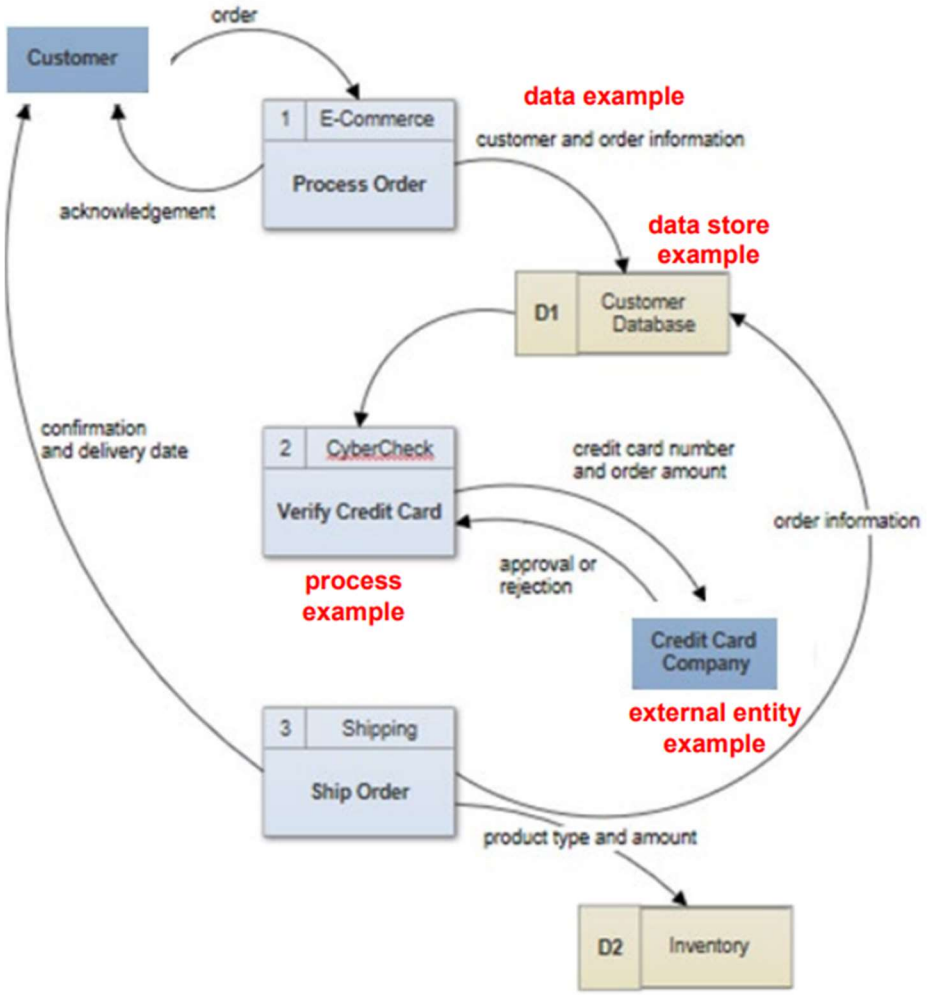
DFD Level 2

- ✓ Breaks down further into parts of Level 1 to reach the **necessary level of detail** about the functions of the system.

Example 2

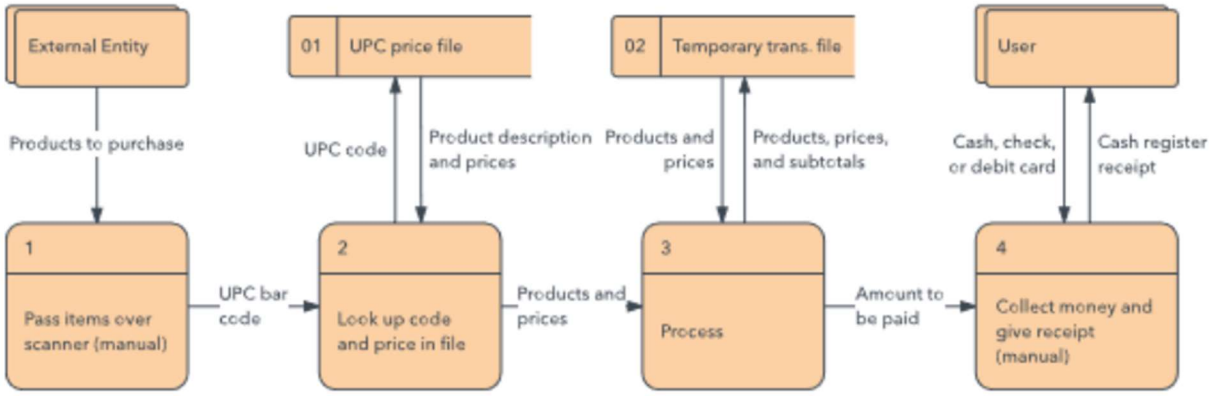


Business Process 1 example:



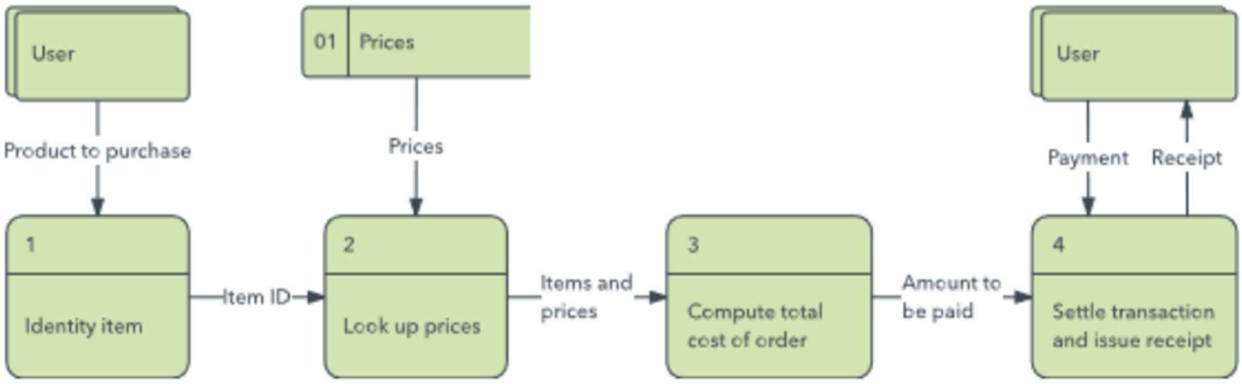
Physical DFD Diagram

- ✓ examines “how” a system is implemented
- ✓ provides “how” the information flows for a process or system
- ✓ describes “how” the data system will work, such as the hardware, software, files and users involved.



Logical DFD diagram

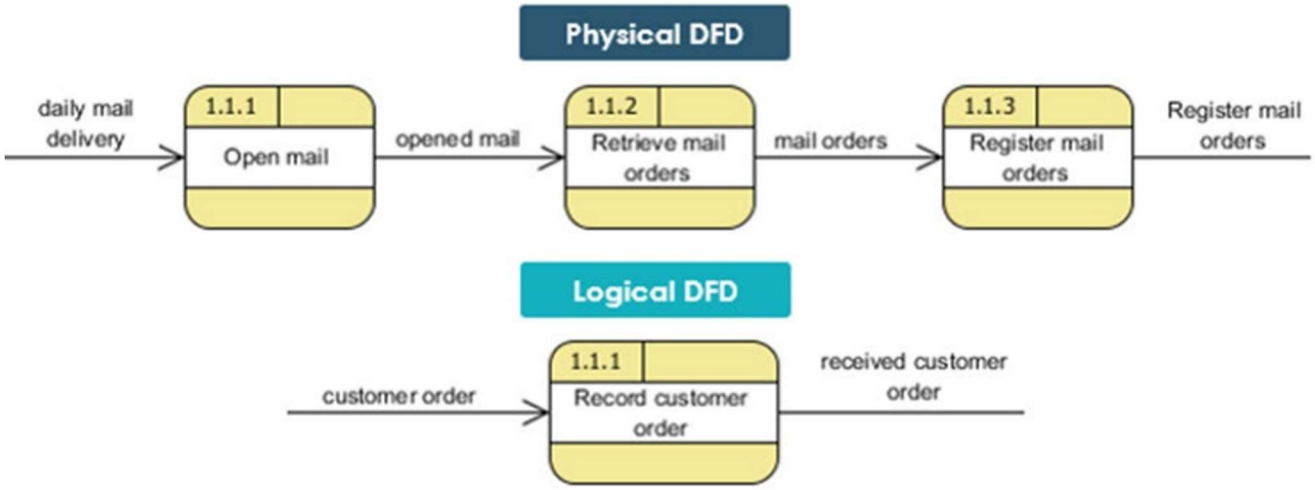
- ✓ provides "what" is in the flow of information for a process or system
- ✓ describes the **events** and the **data** required for each event.



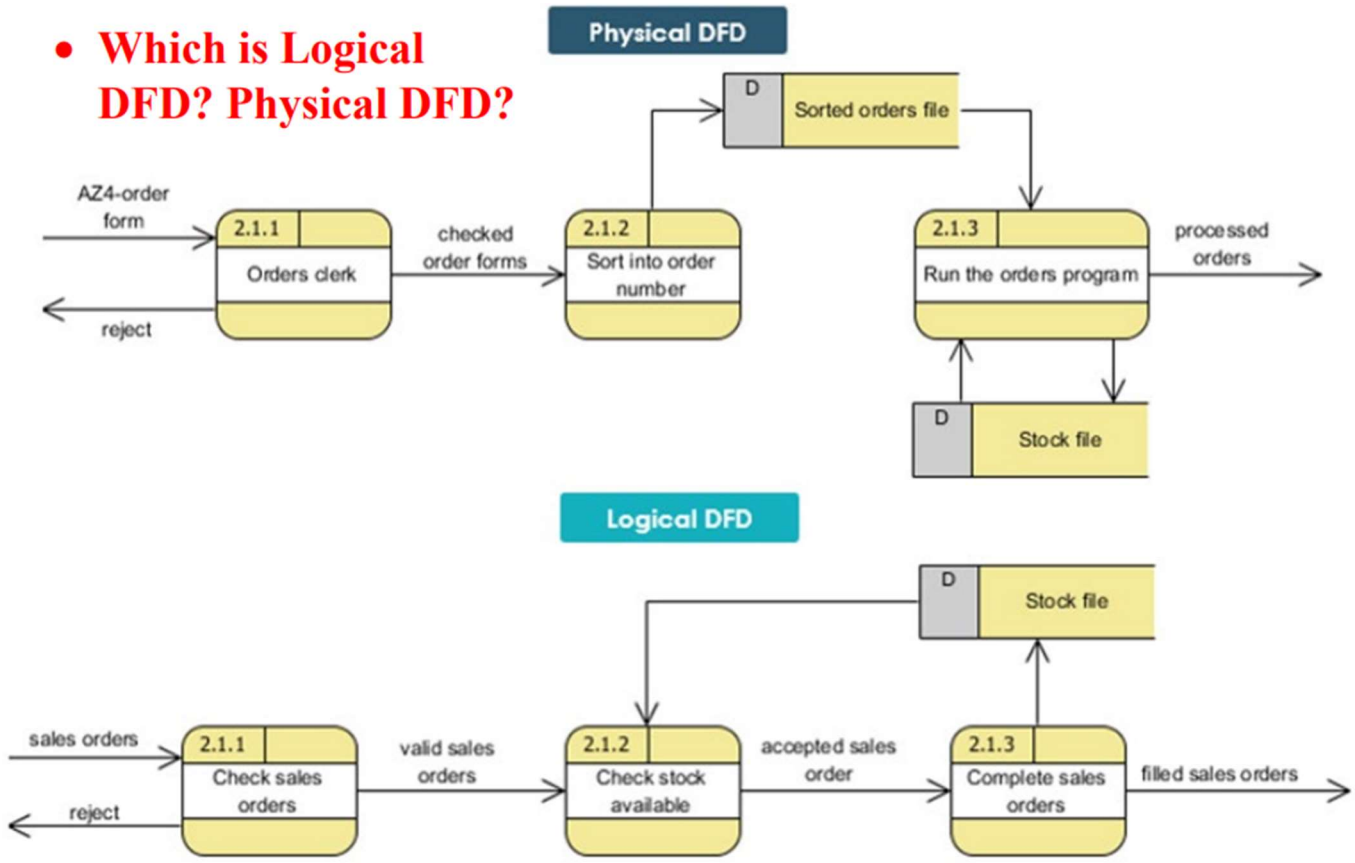
Differences between logical and physical DFDs:

1. **logical DFD describes the flow of information from the perspective of the user/business activities;** physical DFD represents software programs, and consider how the information flow would be implemented.
2. **logical DFD's data stores are collections of information;** in physical DFDs, data stores are databases, computer files and paper files.

Logical vs. Physical DFD: Example



Example 2



Flowchart (and its notations)

− Used to model business processes or software programs

Notations

Start or Stop	Start/Stop	The beginning and end points in the sequence.
Process	Process	An instruction or a command.
Decision	Decision	A decision, either yes or no. For example, a decision based on temperature that turns a central heating system on or off.
Input or output	Input/Output	An input is data received by a computer. An output is a signal or data sent from a computer.
Connector		A jump from one point in the sequence to another.
Direction of flow		Connects the symbols. The arrow indicates direction.

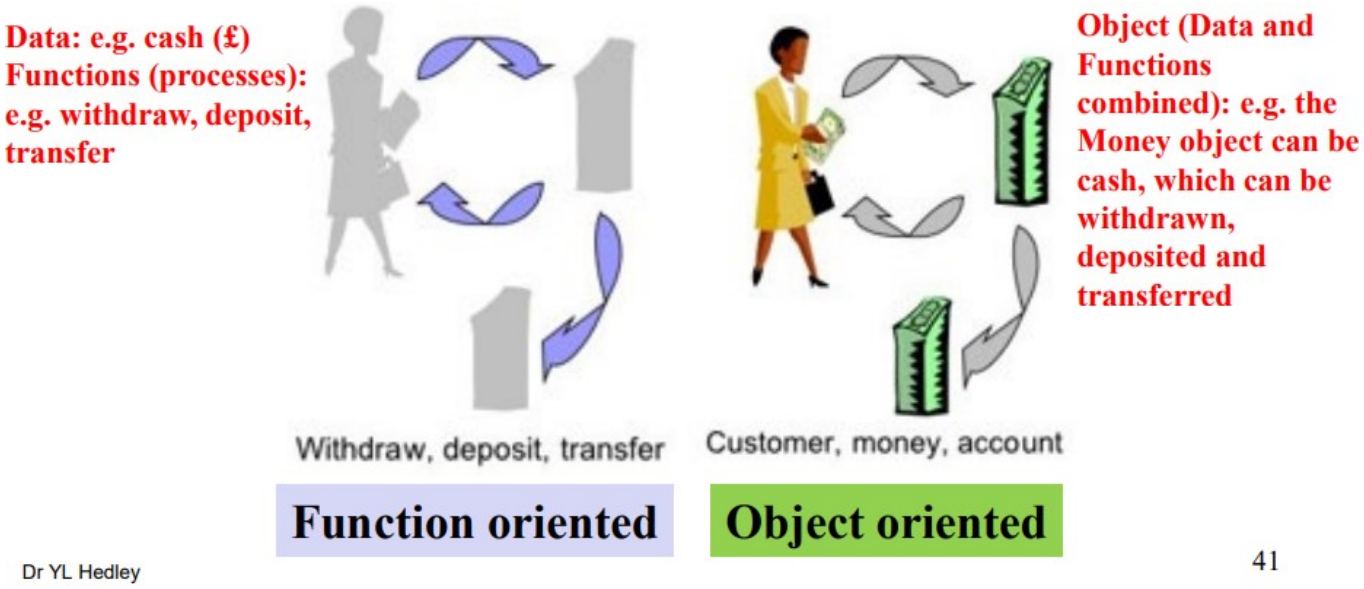
Data Flow Diagram vs. Flowchart

Key differences:

- ✓ DFD presents the flow of data that is transformed by the process; FC presents the steps to complete a process
- ✓ DFD describes the flow between external source and internal store; FC does not describe input from or output to external sources
- ✓ DFDs shows the flow of data; FC shows the flow of control

Object-orientated (vs. function oriented)

Object oriented is based on the concept of objects in which data is encapsulated with the functions that act on the data.



Object-Oriented Approach:

- ✓ Structural Design
  - Focuses on **static** aspects of the software system
  - Techniques: e.g. class diagram, object diagram, component diagram, etc.
- ✓ Behavioural Design
  - Focuses on **dynamic** aspects of the software system
  - Techniques: e.g. activity diagram, sequence diagram

Week 4: Relational Database & Modelling

What is database? a collection of related data, which can be processed to produce information. **Example**: database of students' records

What is Database Management system (**DBMS**)? A software system which can create and manipulate databases as well as store and retrieve data (**Examples**: MySQL, Oracle, SQL-Server)

DMBS types:

- **Relational**: holds tables of data
- **Object**: holds objects (containing attributes)
- **Object-Relational**: combines abilities of relational and object oriented databases
- **NoSQL** ("Not only SQL"): consistent, capable of storing a huge amount of data.

DBMS system has **ACID** properties!

a) **Atomic** - Once started, either all its tasks are done (committed) or all its tasks are undone (rolled back).
b) **Consistency** - If a database is consistent before executing a transaction, it will also be so afterward. In other words, after performed action, you cannot change it anymore
c) **Isolation** - this ensures that all transactions will occur in isolation. That is, the order of their executions (and therefore their concurrency) is irrelevant to their final effect on the database.
d) **Durability -** this ensures that once a transaction is committed, it will remain in the system. Any changes from the transaction must be stored permanently.

Main **DBMS features:**

- Isolation of data and application
- **Multiple views:** DBMS offers multiple views for different users, who have a concentrate view of the database according to their requirements.
- **Security:** e.g. through multiple views to enable users to have different views with different features.

Database Schema

Defines its entities and their relationship along with the constraints applied on the data, in two categories:

- **Logical database schema**: defines logical structures and constraints to be applied on the data store, including tables, views, and integrity constraints.
- **Physical database schema**: concerning the actual storage of data in a secondary storage

Main points to remember about **Relational data model**:

- ✓ Data are stored in **tables** also called relations
- ✓ Each row in a table is called a **record** or a **tuple**.
- ✓ Each column is called an **attribute**.
- ✓ **Domain**: a set of acceptable values for a column
- ✓ **Degree**: is the number of attributes in a table.

Relational Database uses the relational data model for data storage and processing:

1. Relations can be **normalized**, which reduces redundancy (will get back to this later on today)
2. Each row in a relation contains a unique identifier called **Primary Key** (PK), E.g. SID (Student ID)
3. **Structured Query Language (SQL)** is used as the standard database access language

Entity Relationship (ER) data model, also called an ER schema! ER model describes: **Entities** and **Relationships** among entities

ER model is presented by **Entity Relationship Diagram (in those notations)**

a) Chen notation
b) Crow's Foot notation

ER model:

• **Entity**: An entity can be a real-world object, such as student, course, tutor

• **Attributes**: represents the properties of an entity. Each attribute has a value, e.g., a student entity may have a name as an attribute with a value, John.

Cardinality - defines the number of entities, which can be associated with the number of other entities via a relationship in this way: One-to-one, One-to-many, Many-to-one, Many-to-many

Degree of Relationship: the number of participating entities in a relationship

1 – Unary, 2 – Binary, 3 – Ternary, n-nary.

Entity Relationship Diagram (ERD) **types**:

- **Conceptual** used by business analyst
- **logical** used by database designer to design the database but at higher level
- **physical** used by database designer to improve the logical design and prepare for actual database construction.

Conceptual ERD:

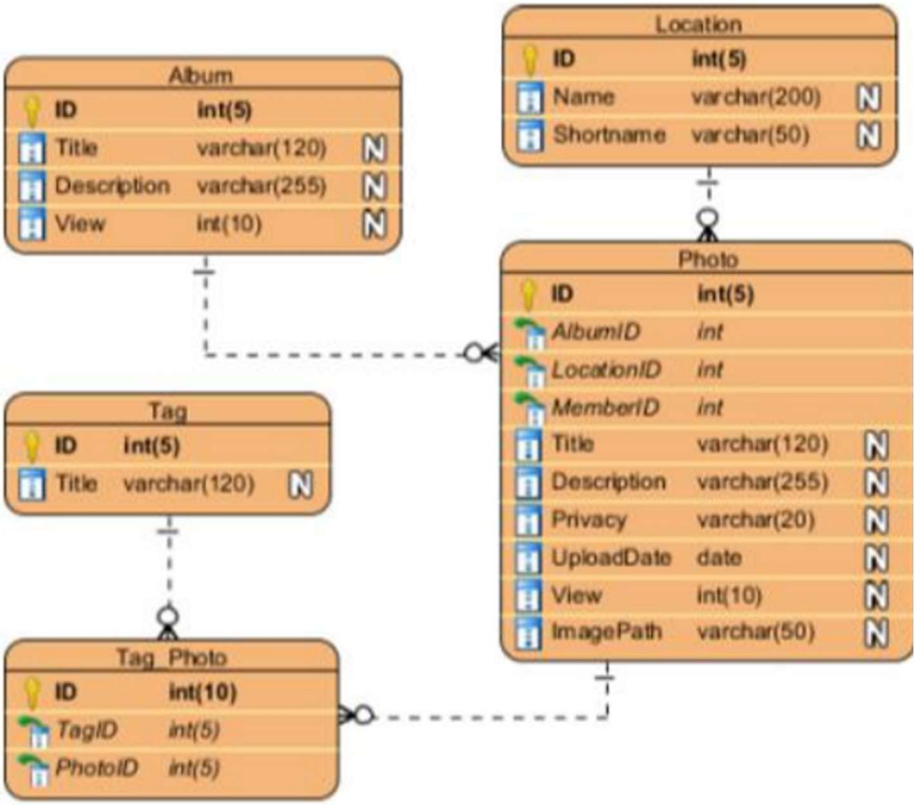
- models information gathered from **business**. Without detailed structure such as data types
- created by business analyst, product owners and other stake holders

Logical ERD:

- also models information gathered from business **but setting data types**
- defines the detailed structure of the entities and their relationships

Physical ERD:

- considers platform specific convention and restriction and ensures primary keys, foreign keys and constraints to the design
- **visually represents the *ACTUAL*** database schema
  - **Keys**: used to link various tables in a database
  - **Candidate key**: a simple or combined key that is unique.
  - **Composite key**: is made of two or more attributes, but it must be minimal.
  - **Primary Keys**: identifies a one tuple
  - **Foreign Keys**: enable an attribute to relate to another entity in a **one-to-one or one-to-many** relationship.



ERD example 1

Database Design Strategies

a) **Top down** approach: To start from the general understanding of what is needed for the system, and to progress to more specific details of how the system will interact, including the identification of different entity types and the definition of each entity's attributes.
b) **Bottom up** approach: To start with the specific details and move up to the general. That is, to identify the data items, the attributes, and then group them to form entities

Normalisation - the process of removing redundancy in a table

Forms can be: First normal form(1NF), 2NF, 3NF...

1NF: Each cell of the table has an atomic value, as opposed to a list of values

For example, if we want to keep information about all courses a student passed in the following table, the table will NOT be in 1NF:

Student_Grade_Report (StudentNo, StudentName, Major, CourseNo, CouseName, InstructorNo, InstructorName, InstructorLocation, Grade)

So, what to do ? Remove such repeating groups by splitting the table into two as follows:

Student (StudentNo, StudentName, Major) → StudentCourse (StudentNo, CourseNo, CouseName, InstructorNo, InstructorName, InstructorLocation, Grade)

Note the line under the unique identifier, i.e. primary key (PK), in each table

2NF: To move to 2NF, a table must first be in 1NF

The Student table is already in 2NF because it has a single-column PK.

Normalisation: 2NF. In StudentCourse table, all course information are not fully dependent on the PK, because courseNo can uniquely identify it.

So, we put all course information in a new table with courseNo as its PK:

Student (StudentNo, StudentName, Major)

CourseGrade (StudentNo, CourseNo, Grade)

CourseInstructor (CourseNo, CouseName, InstructorNo, InstructorName, InstructorLocation)

Normalisation: 3NF. So, we further split it into two other tables:

Student (StudentNo, StudentName, Major)

CourseGrade (StudentNo, CourseNo, Grade)

Course (CourseNo, CouseName, InstructorNo)

Instructor (InstructorNo, InstructorName, InstructorLocation)

Finally! Now, every non-key attribute is functionally dependent on PK only

Week 8: GoF Design Patterns and Ethics

'Gang of Four' analysed 23 Design patterns which provide solutions to general problems faced during software development as follows.

- **Creational** patterns - manage the creation of objects
- **Structural** patterns - describe how objects are connected together to form more complex objects
- **Behavioural** patterns - describe how code is organized, to assign responsibility or roles to certain classes, and to specify the way objects **communicate** with each other

Creational	Structural	Behavioral
• AbstractFactory • Builder • **FactoryMethod** • Prototype • **Singleton**	• **Adapter** • Bridge • Composite • **Decorator** • Facade • **Proxy**	• TemplateMethod • Visitor • Command And more...

Creational patterns:

• separates a system from the creation, composition and representation of its objects , which increases the system's flexibility in what, who, how, and when of object creation.

• encapsulates the knowledge about which classes a system uses and hides the details of how the instances of these classes are created and structured.

1. Factory Method pattern

a. used to create objects, but allow subclasses to decide exactly which class to instantiate with various subclasses implementing the interface
b. instantiates the appropriate subclass based on information supplied by the client or extracted from the current state.
c. is useful when requiring the **creation of many different types of objects**, all derived from a common base type.
d. – defines a **method for creating the objects**, which subclasses can then override to specify the derived type to be created.
e. – At run time, can be passed a description of an object and return a **base class pointer to a new instance** of that object.
f. Requires a well-designed **interface** for the base class, instead of casting the returned object (e.g., in Java)
g. To support additional object types – when an additional class is required, and objects are requested through a user interface, this pattern would simply pass on the new information to the factory, which would then handle the new types entirely.
h. An object is created without exposing the creation logic to the external using a common interface

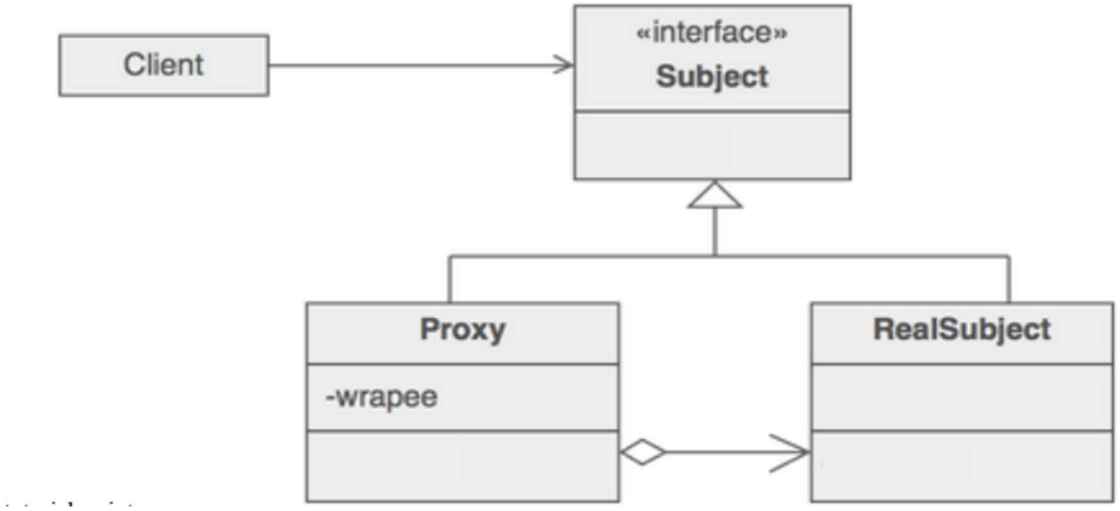
2. Singleton

a. enable a single class responsible to creates own object while ensuring that only **single object** get created
b. provides the **access** to its only object **directly** without need to instantiate the object of the class.
c. **useful** when exactly **one object** is **needed** to coordinate across the system
d. Usually used for centralized management of internal or external resources to provide a **global point of access** to themselves.
e. should ***avoid*** introducing unnecessary restrictions in situations where a sole instance of a class is not actually required, and also introducing global state into an application
f. Singleton pattern: implementation
   i. Define a private **static attribute** in the "single instance" class.
   ii. Define a **public static accessor method** (i.e. get method) in the class
   iii. Define all **constructors** to be protected or **private**
   iv. **Use** only the **accessor method** to manipulate the Singleton
g. **Example**: the singleton pattern is applied to provide the global access to a file system

Structural patterns:

1. Proxy Pattern

a. – provides an object as a placeholder for another object to control access to it, including:



b. **Remote Proxy:**

i. provides **a local representation of the object** which is present in the different address location

ii. provides the **interface for remote resources** such as web service resources.

iii. ***Example***: When an object and its methods is running on another computer, and cannot be called directly

> **Solution**: To open a socket on the remote machine and pass messages to the remote object via a protocol as if the object was local. The methods can be called on a proxy object that forwards the calls to the real object on the remote machine, such as in CORBA (Common Object Request Broker Architecture), in ASP.NET (Active Server Pages for .NET), and in Java's Remote Method Invocation (RMI).

c. **Smart Proxy:**

i. provides additional layer of security by interposing some actions when the object is accessed.

ii. ***Example***: a proxy to check if the real object is locked before it is accessed to ensure that no other object can change it.

d. **Protective Proxy**

i. ***Example***: A company imposes a policy that employees will now be prohibited internet access based on their roles. All external emails websites will be blocked.

> Solution:
> ♣ InternetAccess interface which consists of operation grantInternetAccess().
> ♣ RealInternetAccess which allows of internet access for all.
> ♣ ProxyInternetAccess which restricts the internet access, which will check user's role and grant access based on their roles.

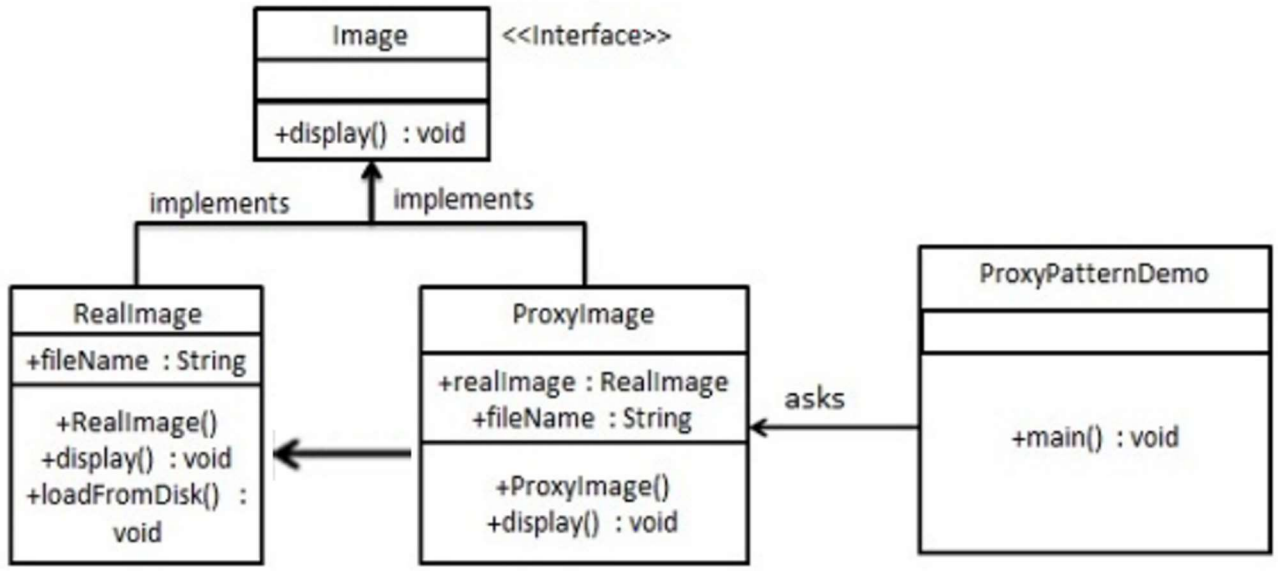
e. **Virtual Proxy**

i. useful to **save expensive memo**ry resources when there is an expensive operation, multiple proxies can be created and pointed to the huge size memory consuming object for further processing.

ii. The real **object gets created only when a client first requests**/accesses the object and after that, the proxy object can be referred and reused. This avoids duplication of the object and hence saving memory.

iii. ***Example***: a real image contains a huge size data which clients needs to access.

Solution:
♣ Image interface which has operation display().
♣ RealImage runs on the different system and contains the image information is accessed and loaded from the database.
♣ ProxyImage which is running on a different system can represent the RealImage in the new system. Using the proxy, multiple loading of the image can be avoided.



Virtual proxy example 1

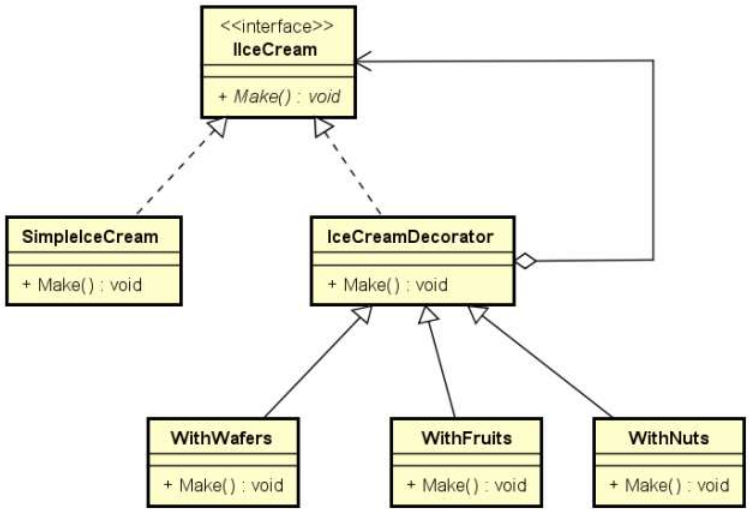
Proxy example: Example: the Processor has RAM with cache memory; the proxy pattern provides an proxy object (i.e. Cache) to the real object (i.e. RAM) for the memory access

2. Adapter pattern

a. works as a **bridge** between two incompatible interfaces
b. merges two independent interfaces
c. uses a **single class** which is responsible to **join functionalities** of interfaces
d. ***Example***: card reader which acts as an adapter between memory card and a laptop. A memory card can be plugged into card reader and card reader into the laptop so that memory card can be read via laptop.

3. Decorator Pattern

a. allows a user to **add** new **functionality** to an existing object without altering its structure.
b. acts as a **wrapper** to existing class.
c. creates a decorator class which wraps the original class and provides additional functionality keeping class methods intact.



Example decorator 1

ETHICS ISSUES: Examples

- ✓ **Privacy**: storage, sharing of user data only for the purposes that the user sets
- ✓ **Transparency**: relating to transparent decision-making procedures of intelligent systems, publicly available ethics policies by software development organizations
- ✓ **Diversity**: take into account of gender, race, and age distribution of professionals in a development team
- ✓ **Work ethics**: relating to decisions on which bugs to fix and how quickly, ensuring quality of the code before release
- ✓ **Accountability**: concerning responsibility for the harm caused by software
- ✓ **Dependability**: concerning the maintenance and keeping of a software product in the market
- ✓ **Encryption**: to prevent maliciously accesses the private information and to provide due diligence toward protecting information by the software development organizations
- ✓ **Intellectual property and copyright**: Intellectual property results in copyright laws and licence agreements

PROFESSIONALISM:

- Professional bodies are the guarantors of your professionalism relating to the responsibility and trust which can be placed on you
- In the United Kingdom the **British Computer Society (BCS) for computing professionals** exists

Poor Professional Conduct: ***Examples***:

- ➢ **Blind Coding**: implementing a bug fix without testing it
- ➢ **Death March**: a project known doomed to fail but is ordered to keep working on anyway
- ➢ **Lava Flow**: old code, often undocumented and with its function poorly understood, thus left in
- ➢ **Software Bloat**: successive iterations of software using more and more resources with little or no added functionality
- ➢ **Spaghetti Code**: unstructured, messy code not easily modified or extended
- ➢ **Error Hiding**: overriding the display of error messages with exception handling, so cannot be seen neither by the user nor tech support