A2 SSD

Contents

[Reasons for Systems Development 3](#_Toc491194705)

[Reasons for Initiating a Project 3](#_Toc491194706)

[Projects are initiated in an organisation for various reasons: 4](#_Toc491194707)

[Why are new systems developed? 4](#_Toc491194708)

[Components of IT Systems 4](#_Toc491194709)

[Main Stage of Developing an IT System 4](#_Toc491194710)

[Analysis 5](#_Toc491194711)

[The Analysis Phase 7](#_Toc491194712)

[Design Phase 8](#_Toc491194713)

[Importance of Successful Analysis and Design Phase 8](#_Toc491194714)

[A Feasibility Study May Be Carried Out 8](#_Toc491194715)

[Testing 9](#_Toc491194716)

[Integration 10](#_Toc491194717)

[Maintenance 11](#_Toc491194718)

[Roles in a Project 12](#_Toc491194719)

[User 12](#_Toc491194720)

[Analyst (Systems Analyst/Business Analyst) 12](#_Toc491194721)

[Programmer 13](#_Toc491194722)

[Project Manager 13](#_Toc491194723)

[Systems Methodologies 14](#_Toc491194724)

[Iterative Methodologies (Agile) 14](#_Toc491194725)

[Rapid Application Development (RAD) 14](#_Toc491194726)

[Dynamic Systems Development Method (DSDM) 16](#_Toc491194727)

[SCRUM 17](#_Toc491194728)

[XP – Extreme Programming 20](#_Toc491194729)

[Traditional Methodologies 21](#_Toc491194730)

[Waterfall Model 21](#_Toc491194731)

[Prototyping 22](#_Toc491194732)

[What is a Prototype? 22](#_Toc491194733)

[Two Types of Prototyping 22](#_Toc491194734)

[Advantages of Prototyping 22](#_Toc491194735)

[Disadvantages of Prototyping 22](#_Toc491194736)

[Traditional Methods Pros & Cons 23](#_Toc491194737)

[Iterative Model Pros & Cons 23](#_Toc491194738)

[Agile Model Pros & Cons 24](#_Toc491194739)

[UML – Unified Modelling Language 25](#_Toc491194740)

[Goals of UML 25](#_Toc491194741)

[Types of UML 25](#_Toc491194742)

[Object Orientated Concepts in UML 25](#_Toc491194743)

[Structural Diagrams 26](#_Toc491194744)

[Behavioural Diagrams 29](#_Toc491194745)

[Basic Notation of UML 32](#_Toc491194746)

[Structural Things 32](#_Toc491194747)

[Behavioural Things 35](#_Toc491194748)

[More on Use Case Diagrams 39](#_Toc491194749)

[More on Class Diagrams 39](#_Toc491194750)

[More on Sequence Diagrams 40](#_Toc491194751)

[Database Concepts 40](#_Toc491194752)

[What is a File Processing System? 40](#_Toc491194753)

[Characteristics of the File Processing System 40](#_Toc491194754)

[Advantages 40](#_Toc491194755)

[Disadvantages 40](#_Toc491194756)

[What is A Database? 41](#_Toc491194757)

[Characteristics of a Database 41](#_Toc491194758)

[Advantages Of Databases 41](#_Toc491194759)

[Disadvantages of Databases 41](#_Toc491194760)

[Advantages of Database Approach to Traditional File Approach 42](#_Toc491194761)

# Reasons for Systems Development

* analyse the reasons for project initiation and systems development within organisations;
* examine and apply analysis and design principles to business solutions;
* justify the roles inherent in the systems development process, particularly the role of the user;

## Reasons for Initiating a Project

Organisational growth

Problems with existing system

Desire to exploit new opportunities

Change in market or external environment

Desire to make more effective use of information

Increasing competition

Merger or acquisition

Reasons for Initiating a Project

## Projects are initiated in an organisation for various reasons:

* **Market Demand**
* **Strategic Opportunity or Business Need**
* **Customer Request –** if a customer has a request for something to be done, it forms a project
* **Technological Advance**
* **Legal Requirements**
* **Social Need**

## Why are new systems developed?

* Current system is out of date and technology has moved on
* Competitors developed new systems
* Organisation has grown. A new part of the company has been established that requires IT support.
* Company wants to improve the quality of a repetitive task.

## Components of IT Systems

* People
* Data
* Procedures
* Software
* Hardware
* Information

# Main Stage of Developing an IT System

This is known as the systems development life cycle (SDLC). There are 7 main stages:

* **Analysis** : precisely describing the problem in order to establish the user’s requirements or needs, as well as suggesting possible solutions
* **Design** : describing in detail how the chosen solution will function
* **Implementation (software development):** turning the design into an actual working system
* **Testing:** ensuring the system works as expected
* **Installation (integration):** the new system is installed and end users put the system to use
* **Review (evaluation):** of the system once customers have had a chance to use it
* **Maintenance:** after implementation the system may continue to develop as new features are added (adaptive), bugs are corrected (corrective) or is made to work more efficiently (perfective)

At the end of each stage, agreed deliverables must be produced which are signed off by the client together with approval by the project manager to proceed to the next stage.

The purpose of the SDLC is to maximize the chance of a successful project.

(SDLC sometimes given as: planning, analysis, design, implementation, testing, integration, maintenance.)

## Analysis

Analysis involves detailed examination of any current system to establish user requirements. It is possible that a system that was once perfectly adequate has become outdated due to changes in working practice, so no longer meets the customer’s requirements. In this case, the new system may be expected to do everything its predecessor did as well as many new tasks. An existing system may be paper based or run on an obsolete platform.

This phase is carried out by a system analyst (or business analyst) who determines

1. User Requirements
2. Any failings of the current system
3. Input data and its source.
4. Output data and its destination
5. Constraints on hardware, software and development time
6. Whether what the customer wants is what the customer needs
7. Whether any proposed solution is feasible in reality.

The analyst investigates the customer’s current system using several different fact-finding methods.

Staff Interviews

* Talking on a one-on-one basis with a broad spectrum of people within the organisation, from management to rank and file personnel, and seeking their opinions
* The views of management are highly important as they ensure any solution will address company-wide issues.
* Other personnel may be expected to use a system daily, so failure to consider their concerns may lead to an inoperable system which will lead to discontented staff and reduced productivity,

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Flexible enough to be tailored to each person * Allows follow up questions to individuals * Facts are gathered from system users, a direct source | * They require careful preparation as open and closed are needed * Can be costly and time consuming * Employees may feel uncomfortable or threatened about why they are being questioned about the system and fail to give full answers |

Examining Current Business Documentation

Any paperwork entering the existing firm, and current management reports

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Help to identify data requirements and current input and output for the existing system * Includes both internal (inside system) and external data flows (outside system). | * Studying documents is time consuming and tedious * Does not give feedback or views from system users |

Questionnaires

When interviews are impossible for every employee, questionnaires may be used. They must be carefully written so answers given are relevant, clear and ambiguous. Consequently, multiple choice forms are often used.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * When used well, they can gather information from a lot of people over a large area and are relatively cheap to organise | * There is no way to tell how truthful a respondent is being |

Observation

Carefully watching how the current system is used in a typical working environment and understanding in depth how it is used.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Data collected is very accurate in nature and very reliable * Analyst can identify a problem by making an in depth analysis of the problems | * Opinions of the users cannot be studied |

## The Analysis Phase

Represents the ‘what’ phase. It defines the problem that the customer is trying to solve. The deliverable at the end of the phase is a requirements document. There are 3 steps:

1. Analysis Strategy
2. Requirements
3. Gathering

Analysis Strategy

Details the problems of the current system. Then proposes a better way of achieving the same task by way of a software application.

Requirements Gathering

Defines the requirements of the proposed system, regardless of how they will be accomplished, the end goal being a requirements document.

This document states in a clear and precise manner what a system should do when it has been built.

It does not describe how the system will be built at an architectural level nor will it detail anything about the code to be used or how the code will be written. It will give a high level view, specifically the problems to be solved, expectations of the customers and end users, and also the criteria for success – what must the project achieve to be considered successful to the customers specification.

There are cases of exception regarding the exclusion low level requirements from a requirements document:

* If the proposed system is to be built upon or integrated with an existing system
* If off the shelf software is to be used (another system to be purchased to use in conjunction)
* Memory constraints restricting the size of the application

These low-level details can be included. It is considered a flexibility of the document and not a rule, it is the authors discrepancy and not a rule if they are included or mentioned in a general sense.

Traditionally the document describes ‘things in’ the system and the actions that can be done on these things. During the design phase these ‘things’ will be objects and actions will be methods associated with them.

System Proposal

Gathers all the information about the new system into a document and presents it to stakeholders and then a decision is made by them whether to move forward with the project or whether a rewrite is needed to change requirements to suit their expectations.

A poorly undertaken analysis phase will lead to a poorly implements application and an unhappy customer.

## Design Phase

Represents the ‘how’ phase.

Using the documentation to map the requirements into an architecture. The deliverable is the system specification, which fully describes the systems architecture.

The System Specification Will Contain:

* An overall, high level design of the system giving an overview of the entire application.
* Determine what software and hardware they will use, such as servers for hardware and specific software technologies and languages to be used.
* Program design, this is a breakdown of the system into manageable modules. It helps make future changes easier to implements because it is common in a software project that changes are made. Modules also assist in efficiency and consistency; a well-designed module can be reused in multiple parts of the system saving time and effort and ensuring the same feature of a system looks and performs the same way throughout the system.
* Details of interfaces necessary between the new software system and any existing system it will integrate with.
* A resource estimate to determine how much time and budget that will be required.

## Importance of Successful Analysis and Design Phase

They determine what will be produced at the end of development.

Analysis decides what the system will do and design will decide how the system will do it.

A risk or problem that arises at this stage will not be too costly to fix. If not discovered here it will become costlier to fix as the project progresses.

## A Feasibility Study May Be Carried Out

A feasibility study is a tool for determining whether you have what it takes to undertake a change or start a new project. A project manager may do this.

There are 5 key areas (TELOS):

* Technological
* Economic
* Legal
* Operational
* Scheduling

Technological

Can you build it? It you can’t, can you find and pay for experts who can? If it were built, would you know how to use it?

Economic

Where is, the money coming from to fund the project? Will upper management support the project to a degree that they will make funding available? Given the costs, is the money relatively l=high compared to other high priority projects?

Legal

Ensuring the project does not break any laws e.g. patent or trademark laws, existing contracts, confidentiality agreements. Data Protection Act, Computer Misuses Act, Digital Economy Act, Copyright, Designs and Patent Act.

Operational

Will you need to institute new procedures to make this new system successful? What kind of training do people need? What will it cost? Will the project have an operational impact on people outside of your own team?

Scheduling

Relates to the amount of time required to plan, build, train for, and implement a new application.

The amount of extra time that might be involved during development and implementation of a time sensitive client project. Given a realistic look of the time it will take to make and the challenges involved, is it worth it?

## Testing

To ensure effective testing takes place, a test plan created as part of the design process will be used. It will identify the purpose of each test, method of testing, the data to input and the expected output.

* **Unit Testing** During development, this ensures each component functions properly when completed. Leads to **integration testing** when components are brought together once they are known to work independently.
* **System Testing** (or application testing) is carried out to ensure the complete system works according to the specification. Data from current system is put into new system to mimic realistic demands and to highlight any serious flaws without corrupting user data. Ensures that all modules work together.
* **Acceptance Testing** The system is given to the end users to operate in its final environment. It is tested with real data and normal daily workloads. Users can assess whether it meets their expectations. Minor modifications may result until there is an agreement that the system meets the users agreement.

Testing should ensure that:

1. The data the system was built to operate on is processed normally.
2. Unexpected inputs do not crash the system.

To cover these scenarios, 3 types of input data are used.

1. **Reasonable, correct data.** Normal data the system was built to process
2. **Extreme, correct data.** Data is correct but unlikely to be encountered every day.
3. **Incorrect data.** Data that is erroneous and should not be accepted.

## Integration

Once the system in fully developed it is put into use on the customer premises. This may lead to problems that must be overcome.

* New hardware may have to be purchased, installed and tested. Technical staff may have to be hired and paid to install the new system.
* Training may have to be provided to all staff, which can be costly. Training must suit the needs of all staff intending to use the new system. It is also time sensitive; staff must be ready for implementation of the new system.
* Data held on old system must be imported to new system with complete accuracy. Data conversion tools may have been created to automate the process.
* Employees may not want to change from systems they are used to. Redundancy payment may be offered to staff who cannot adapt to the new system.

The changeover method must be considered carefully. There are 4 main types:

Parallel Changeover

The new system runs simultaneously with the old for given period. All users duplicate all work on old and new systems for a certain period. When the new system meets all requirements, the old system is dropped. This may take several months.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Most popular method – it carries the lowest risk. If something goes wrong, the entire system can be reverted to its original state * Accuracy of new system can be tested by comparing its output with the output of the original | * Running two systems at the same time so higher costs * Time consuming as users must duplicate all their efforts * Strain on staff – overtime or employment of extra staff |

Direct Changeover

Organisation stops using old system one day and starts using the new system the next day. Tends to be the least favourite technique. The entire system is replaced in an instant. As soon as the new system is powered up, the old system is shut down.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Tends to work best in situations where a system failure isn’t critical enough to result in a disaster for a company * Only one system is operational at a given time so no time is wasted from running two systems * Inexpensive if works seamlessly as extra staff are not required | * Carries the most risk * If something goes wrong the reverting to the old system is usually impossible * Business can be seriously disrupted is the new system is found to have bugs or doesn’t work as expected |

Pilot Changeover

A select group of end users access the new system first. Any errors can be dealt with before the new system is made available to all users.

Training can then be modified based on the experiences of pilot (beta) testers.

The new system is usually tried out at a test site before launching it company-wide.

Parallel changeovers tend to be expensive, using pilot changeover technique allows companies to run the new system next to their old system but on a much smaller scale, making the pilot changeover method much more effective.

After the kinks are worked out at the test site, companies usually opt to use the direct changeover technique to launch the system company-wide.

Phased Changeover

Occurs in stages that can be implemented separately at different times. Old and new systems must be able to share data as both will be used together for some time.

Phased changeover technique is considered a compromise between parallel and direct changeover. The new system is implemented one stage at a time.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Low cost * Isolated errors * Different users will be expected to use different parts of the system. Once a significant number of people have started using the new system, most of the serious problems have been discovered and fixed or users know how to overcome them. Process is easier for significant groups. | * Process takes a long time to complete because phases need to be implemented separately. * Only suitable for systems that operate as clear distinct modules. |

## Maintenance

Ensures that the system continues to run as expected by being adapted to new requirements. There are three types of maintenance:

* **Corrective** maintenance corrects bugs or remedies aspects of the systems that do not properly meet requirements
* **Perfective** maintenance makes the system work more reliably by removing inefficiencies and improving performance
* **Adaptive** maintenance adds new functionality to the system because of changing business needs or factors outside of the business’s control. Could be required to take advantage of hardware or to respond to changes in a system’s environment, such as new tax legislation or competitor’s products.

# Roles in a Project

Roles within a development team include; user, analyst, programmer, project manager, designer, and tester.

## User

A user is someone who uses the computer system daily. Users represent those people who are going to be working with the system on a regular basis. Users can be at any of the level of the organisation, from employees on the work floor to senior executives – this depends on the nature of the system being developed.

During analysis phase, the analysts will gather information to build a set of requirements which will be necessary for the system to be successful to the client.

The user generally has little input during the design phase as it is a technical document describing software structure.

During implementation phase, they act as the last resort when it comes to a further question on the business logic.

Once testing is underway the stakeholder becomes the final tester. This is user acceptance testing.

The stakeholder will use the application and check that it is doing all the functions correctly. Important as it ensures the product is doing what the customer described.

## Analyst (Systems Analyst/Business Analyst)

A professional with expertise in analysing and designing business systems. Plays a critical role in understanding how the current systems operate and developing new systems that meet specific objectives. They provide detailed specifications for what the final system is supposed to look like and how it is supposed to work.

Often they are the most central person on the development team and the only person who would see the project through in its entirety. Users focus on how the system will function for them. Programmers are often focused on writing solid code. Managers are focused on making sure the project is on time and on budget. System analysts play a pivotal role by interacting with all team members and coming up with a system design that meets the original objectives, will be enjoyed by users can be built by programmers, and can be accomplished with all the available resources.

What does an analyst do?

* Helps organisation to understand the challenges before them to ensure that the needs and expectations of the client are represented correctly in the final solutions
* Responsible for ensuring that the requirements set forth by the business are captured and documented correctly before the solution is developed and implemented
* Analysing and understanding the current state processes to ensure that the context and implications of change are understood by the clients and project team
* Developing an understanding of how present and future business needs will impact the solution
* Identifying sources of requirements and understanding how roles help determine the relative validity of requirements
* Identifying and documenting all business, product and process requirements
* Working with the client to prioritise and rationalise requirements
* Helping to define acceptance criteria for completion of solution

## Programmer

A technical specialist responsible for developing specific software that meet specifications provided by the analyst. A programmer will implement the system design as specified by the analyst.

Writing code is often one of the more time-consuming parts of system development, especially if many of the components must be custom built. A system design developed by a single analyst may require a few different programmers to develop.

## Project Manager

Responsible for coordinating various team members and resources to complete projects on time. PMs need several different skills, including technical, business and people skills. They need to up to date with different approaches to system development but are not likely to do any programming themselves.

A PM is responsible for controlling the quality of the system being developed, bringing personnel onboard and training them, facilitating communications among the team members and acquiring various resources needed to support the activities of the various members of the development team.

# Systems Methodologies

The development of a project can be categorised into two different methods; Linear and Iterative(agile).

* **Linear Methodologies** – such as the waterfall model are very sequential. They tend to be easy to manage but are inflexible and less able to respond creatively to problems and opportunities that present themselves along the way.
* **Iterative Methodologies** – such as scrum are very agile. Iterative methodologies loop around the stages of development until the developers and/or client are happy with the solution at that point and can move on to the next stage.

# Iterative Methodologies (Agile)

## Rapid Application Development (RAD)

The RAD model is based on prototyping and iterative development with no specific planning involved. RAD focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concepts, reuse of the existing prototypes (components), continuous integration, and rapid delivery.

What is RAD?

A software development methodology that uses limited planning in favour of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

The functional modules are developed in parallel as prototypes are integrated to make the complete product for faster product testing.

There is no detailed planning so it is easier to incorporate changes within the development process.

RAD projects follow iterative and incremental models and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component prototype.

The most important aspect of RAD is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

Phases of RAD Model

* **Business Modelling** – A business model for the product under development is designed in terms of flow of information and distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed, and what are the factors driving successful flow of information.
* **Data Modelling** – The information gathered in the Business Modelling phase is reviewed and analysed to form sets of data objects vital for business. The attributes of al data sets are identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model
* **Process Modelling** – The data object sets defined in the data modelling phase are converted to establish the business information flow needed to achieve specific business objectives as defined in the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying data objects are given.
* **Application Generation** – The actual system is built and coding is done using automation tools to convert process and data models into actual prototypes.
* **Testing and Turnover** – Overall testing time is reduced in RAD as the prototypes are independently tested in every iteration. However, data flow and the interfaces between components need to be thoroughly tested with complete test coverage. Most of the programming components have already been tested, so the risk of major issues is reduced.

RAD Model vs Traditional Model

The Traditional SDLC follows a rigid process with high emphasis on requirement analysis and gathering before the coding starts. It puts pressure on the customer to sign off the requirements before the project starts and the customer doesn’t get the feel of the project as there is no working build available for a long time.

The customer may need some changes after they actually get to see the software, however the change process is quite rigid and may not be feasible to incorporate major changes in the product in the traditional SDLC.

RAD model focuses on iterative and incremental delivery of working model to the customer. This results in rapid delivery to the customer and customer involvement during the complete development of product reducing the risk of non-conformance with the actual user requirements.

RAD Model Application

RAD Model can be applied successfully to projects in which clear modularisation is possible. If the product cannot be broken down into modules, RAD may fail.

Typical scenarios where RAD is used:

* RAD should only be used when a system can be modularised to be delivered in an incremental manner
* If there is a high availability of designers for modelling
* If the budget permits the use of automated code generating tools
* Should only be chosen if domain experts are available with relevant business knowledge
* Where requirements are expected to change over the course of the project and working prototypes are to be presented to the customer in small iterations (eg 2-3 months)

RAD Pros & Cons

* RAD model enables rapid delivery as it reduces the overall development time due to the reusability of the components and parallel development
* Only works well if highly trained engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail

|  |  |
| --- | --- |
| Pros | Cons |
| * Changing requirements can be accommodated * Progress can be measured * Iteration time can be cut short with use of powerful RAD tools * Productivity with fewer people in shorter time * Reduced development time * Increased reusability of components * Quick initial reviews occur * Encourages customer feedback * Integration from very beginning solves a lot of integration issues * Suitable for projects requiring shorter development times | * Dependent on technically strong team members for identifying business requirements * Only system that can be modularised can be built using RAD * Requires highly skilled developers/designers * High dependency on modelling skills * Inapplicable to cheaper projects as cost of modelling and automated code generation is very high * Requires involvement of user throughout the life cycle – they may not be available |

## Dynamic Systems Development Method (DSDM)

An extension of RAD. It was developed to deliver a system that would address the main requirements just as RAD does. Unlike RAD, in DSDM, there must be an efficient and effective layer of communication and cooperation between team members. DSDM is built upon RAD. It seeks to improve it by adding structure and discipline in the aim to increase software quality.

It takes a less rigid approach to functionality, keeping the time of the project fast and fixed and compromising the amount of functionality delivered if problems arise. DSDM is vendor-independent, covers the entire lifecycle of a project and provides best practice guidance for on-time, in-budget delivery of projects, with proven scalability to address projects of all sizes and for any business sector.

DSDM advocates the use of several proven practices, including:

* Facilitated Workshops
* Modelling and Iterative Development
* MoSCoW Prioritisation
* Time boxing

There are usually five main phases:

* **Feasibility Study** – is the project feasible? Is dsdm the right methodology to use
* **Business Study** – looks at current business process through workshops and members of development team. Requirements are built at this point
* **Functional Model Iteration** – Iterations are used to create prototypes which implement major functionality of the software to be produced. Iteration occurs in time boxes.
* **Design and Build Iteration** – the software is developed and finished throughout successive iterations which take place within agreed time boxes
* **Implementation** – users are trained to use the finished software and it’s handed over to the customer

## SCRUM

One of the most popular agile frameworks, which concentrates particularly on how to manage tasks within a team based development environment. Scrum uses and iterative and incremental developmental model, with shorter duration of iterations. It is relatively simple to implement and focuses on quick and frequent deliveries.

**Sprint**

A time- box (2 weeks – 1 month) during which a potentially releasable product increment is created. A new sprint starts immediately after the conclusion of the previous sprint.

**Sprints** consist of sprint planning, daily scrums, development work, sprint review, and sprint retrospective.

In **sprint planning**, the work to be performed in the sprint is planned collaboratively by the **scrum team**.

The **daily scrum** meeting is a 15 minute time boxed event for the scrum team to synchronise the activities and create a plan for the

day.

A **sprint review** is held at the end of the spring to inspect increment and make changes to the product backlog if needed.

The **sprint retrospective** occurs after the sprint review and prior to **sprint planning**. The scrum team inspects itself and creates a plan for improvements to be enacted during the subsequent sprint.

**Roles**

The Scrum Team consists of 3 roles: a scrum master, a product owner and the team.

**Scrum Master**

The Scrum Master is the keeper of the scrum process – responsible for:

* Making the processes run smoothly
* Removing obstacles that impact productivity
* Organising and facilitating the critical meetings

The Scrum Master is a trained, responsible person.

**Scrum Master services to the Product Owner**

* Finding techniques for effective product backlog management
* Helping the scrum team understand the need for clear and concise product backlog management
* Understanding product planning in an empirical environment
* Ensuring that the product owner knows how to arrange the product backlog owner to maximise value
* Understanding and practising agility
* Facilitating scrum events as needed

**Scrum Master services to Scrum Team**

* Coaching scrum team in self-organisation and cross-functionality
* Helping scrum team to create high quality products
* Removing impediments to the scrum teams progress
* Facilitating the scrum team in organisational environments in which what scrum is not yet fully adopted or understood

**Scrum Master services to the Organisation**

* Leading and coaching organisation in scrum adoption
* Planning scrum implementations within the organisation
* Helping employees and stakeholders understand and enact scrum and empirical product development
* Causing changes that increases scrum team productivity
* Working with other scrum masters to increase the effectiveness of the application of scrum in the organisation

**Product Owner**

The product owner is responsible for maximising the value of the product and the work of the team. The product owner is the sole person responsible for managing the product backlog. Product backlog management includes –

* Expressing product backlog items clearly
* Ordering the product backlog items to best achieve goals and missions
* Optimising the value of the work the team performs
* Ensuring that the product backlog is visible, transparent, and clear to all, and shows what the team will work on further
* Ensuring that the team understands items in the product backlog to the level needed

The product owner may do the work or may have the team do it. The product owner remains accountable for these tasks. The product owner is one person, not a committee. The may represent the ideas of a committee in the product backlog, but those wanting to change the priority of items in the product backlog must first address the product owner.

For the product owner to succeed, the entire organisation must respect their decisions. The product owner’s decisions are visible in the content and ordering of the product backlog. No one can tell the team to work on a different set of requirements, and the team is not allowed to act on what anyone else says. This is ensured by the scrum master.

**The Team**

The team is self-organising and cross-functional, comprising of analysts, designers, developers, testers, etc as appropriate and relevant to the project.

Cross functional teams have all the competencies needed to accomplish the work without depending on others not part of the team, thus time and effort can be saved.

The scrum team is designed to optimise flexibility, creativity and productivity.

Optimal team size is small enough to remain nimble and large enough to complete significant work within a sprint. Size should be kept with a range 5 – 9 people, if possible. Fewer than 5 team members decrease interaction and results in smaller productivity gains. Having more than 9 members requires too much coordination.

The team works together closely, daily, to ensure the smooth flow of information and the quick resolution of issues. The team delivers product iteratively and incrementally, maximising opportunities for feedback. Incremental deliveries of a complete product ensure a potentially usefull version of working product is always available.

**Scrum Pros & Cons**

|  |  |
| --- | --- |
| Pros | Cons |
| * Highly visible progress through continual product releases and high involvement of stakeholders * Facilitates change and therefore good for innovative and novel projects where requirements are uncertain * Daily meetings mean issues can be raised at an early stage * High level of communication between stakeholders * Team members productivity is visible | * Requires big commitment from designated users * Intense lifecycle with frequent product releases and change – demanding on team members * Resources required for daily scrum meetings and frequent reviews * Difficult to plan and structure at the beginning as lacks definition – difficult to produce with costs and estimates * Minimal documentation |

## XP – Extreme Programming

XP is based on user stories written by clients as things that the new or updated system needs to do for them with each story being around 3 sentences long.

Acceptance tests are then created to verify that each user story has been implemented correctly

The story developers will receive a detailed description of requirements during implementation and will then work in iterations until the solutions are complete. Paired coding is often promoted.

|  |  |
| --- | --- |
| Pros | Cons |
| * It reduces errors in code as each line of code is reviewed by two people * Accommodates changing requirements through user stories at the beginning of each iteration | * It is code centred rather than design centred. It lacks documentation which can be problematic if members leave and new members join the team * The constant involvement of the client is necessary. This is difficult as the client may not be available or may dislike constant involvement |

# Traditional Methodologies

## Waterfall Model

The waterfall model follows a more traditional approach to software development. It consists of around 7 phases that are followed in prefect sequential order.

1. **Analysis** usually carried out by a systems analyst who will use fact finding methods such as questionnaires, staff interviews, observation of the workplace, and examining current business documentation to find out how the current system works, it’s problems and what the company want from the new system. This information is used to establish the user requirements.
2. **Design** involves describing in detail how the chosen solution will function. The analyst will use the user requirements to produce the new systems designs, which may include data flow diagrams and designs of how the systems should appear to the user. A test plan may also be created at this stage to identify what the system should allow the user to do in more detail.
3. **Implementation** the developers receive the user requirements, the system designs and the test plan and use them to create and develop the new system in small sections (units) at a time. Each unit is developed and then tested for it functionality – this is called unit testing.
4. **Integration and Testing** the entire system is tested using the test plan to ensure effective testing takes place. Each element is tested, the result is documented and failures rectified and retested. This is known as application testing and is carried out by the developers. The system may be given to the end user to assess whether the system meets the expectations of the client. It will test how the system will handle real data and normal daily workloads. The final software which will be set up as the clients’ system may be designed and tested so as there is no problems or confusion during the installation of the product.
5. **Installation** when testing is complete, the system is deployed. Methods of installation include; direct changeover, parallel running, phased implementation, and pilot running. This phase may also include the installation of the new hardware or software, the training of staff so they are able to use the new system and the hiring of technical staff to install the new system.
6. **Review and Maintenance** a review of the entire project will be carried out to determine its success. Strengths and weaknesses will be evaluated and lessons learnt will be recorded to help future projects. Maintenance of the system is carried out to ensure that it continues to run as required by correcting bugs, improving performance and adapting the system to meet new requirements because of changing business needs.

Waterfall maintains the idea that you can only move to the next phase when the preceding phase is completed and perfected. Phases are discrete and there is no overlap between them. The approach is rigid but modifications have been made and new, more agile variations of the approach have risen.

**Advantages of Waterfall**

* The model is very rigid making it easier to manage deadlines
* Works well for small projects where the requirements are understood very well
* Discipline can easily be enforced as each phase can be identified using milestones
* Minimises time wastage and putting emphasis on requirements and design before writing code

**Disadvantages of Waterfall**

* Not appropriate for projects where the requirements have a high risk of changing
* Clients often do not know what they want upfront and prefer repeated two way interactions over the course of the project
* It follows the implicit assumption that designs can easily be translated into real products. Design that looks good on paper can turn out to be difficult in practice
* If in the testing stage, it can be difficult to go back and change something that was not well thought out in analysis or design stages.

## Prototyping

The prototype model provides an alternative to the waterfall model of the systems lifecycle that still produces a quality system without having to identify all requirements in advance.

### What is a Prototype?

A prototype is a working model of a new system used to identify/refine the user requirements to ensure that the system meets the needs of the user. It gives the user useful insight into what is being developed. The user repeatedly evaluates the prototype providing feedback to the analyst so it can be improved and a system which meets the requirements can be developed/produced.

### Two Types of Prototyping

* **Evolutionary Prototyping** – is the building of a working model of a new system which is repeatedly refined based on the users’ feedback until an acceptable system which meets user requirements has been developed. The objective of evolutionary prototyping is to deliver a fully working system to the customer.
* **Throwaway Prototyping** – is the building of a working model which is repeatedly refined until the customer is satisfied that the requirements have been met. This prototype is discarded, it does not become the final system. Only knowledge gained about user requirements and how the system should work is used as the basis of completing and developing from scratch. The purpose of a throwaway prototype is to confirm or clarify system requirements.

### Advantages of Prototyping

* Accommodates client uncertainty well
* Customer sees a partially working version early on
* System can be developed which meets exact needs of the user (direct involvement)
* May reduce development time as inconsistent user requirements will be detected early and resolved. This can reduce costs – changes cost more to implement after they are detected.

### Disadvantages of Prototyping

* If analyst or customer is impatient, design may take place when all the requirements are not clearly known
* Can create situation where the client believes the system is finished (via a good late stage prototype) when only the interface has been developed and coding has not yet started
* Time scale and budgeting may be difficult to manage and control, especially if the user continuously changes requirements
* Less comprehensive documentation is produced and this makes maintenance more difficult

## Traditional Methods Pros & Cons

|  |  |
| --- | --- |
| Pros | Cons |
| * Simple and easy to understand and use * Easy to manage due to rigidity of the model – each phase has a specific deliverable and a review process * Phases are processed and completed one at a time * Works well for smaller projects where requirements are very well understood * Clearly defined stages * Well understood milestones * Easy to arrange tasks * Processes and results are well documented | * No working software is produced until late during the life cycle * High amounts of risk and uncertainty * Not a good model for complex and object-orientated projects * Poor model for long and ongoing projects * Not suitable for projects where the requirements are at a moderate to high risk of changing * So risk and uncertainty is high with this process model * Difficult to measure progress at certain stages * Cannot accommodate changing requirements * Adjusting scope during the lifecycle could end a project * Integration is done as a ‘big bang’ at the very end which doesn’t allow identifying technological or business bottleneck or challenges |

## Iterative Model Pros & Cons

|  |  |
| --- | --- |
| Pros | Cons |
| * Some working functionality can be developed quickly and early in the life cycle * Results are obtained early and periodically * Parallel development can be planned * Progress can be measured * Less costly to change scope/requirements * Testing and debugging during smaller iteration is easy * Risks are identified and resolved during iteration and each iteration is an easily managed milestone * Easier to manage risk-high as risk part is done first * With every increment, operational product is delivered * Issues, challenges & risks identified from each increment can be applied to the next increment * Better risk analysis * Less initial operating time * Better for large and mission-critical projects * During life cycle software is produced early which facilitates customer evaluation and feedback | * More resources may be required * Although cost of change is lesser – not suitable for changing requirements * More management attention is required * System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire lifecycle * Defining increments may require definition of the complete system * Not suitable for smaller projects * More management complexity * End of project may not be known which is a risk * Highly skilled resources are required for risk analysis * Projects progress is highly dependent upon the risk analysis phase |

## Agile Model Pros & Cons

|  |  |
| --- | --- |
| Pros | Cons |
| * Very realistic approach to software development * Promotes teamwork and cross training * Functionality can be developed rapidly and demonstrated * Resource requirements are minimum * Suitable for fixed or changing requirements * Delivers early partial working solutions * Good model for environments that change steadily * Minimal rules, documentation easily employed * Enables concurrent development and delivery within an overall planned context * Little or no planning required * Easy to manage * Gives flexibility to developers | * Not suitable for handling complex dependencies * More risk of sustainability, maintainability and extensibility * An overall plan, an agile leader and agile PM – agile won’t work without it * Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet deadlines * Depends heavily on customer interaction- customer not clear then team is driven in the wrong direction * High individual dependency since minimum documentation is generated * Transfer of technology to new team members may be quite challenging due to lack of documentation |

# UML – Unified Modelling Language

UML is a standardised language for specifying, visualising, constructing, and documenting the artefacts of software systems. It is a pictorial language used to make software blueprints.

## Goals of UML

* To define a general-purpose modelling language which all modellers can use and is also simple to understand
* UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system
* The system can be software or non-software. UML is not a developmental method but it accompanies other processes to make a successful system.

Links for Referencing

<https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language>

<https://www.smartdraw.com/uml-diagram/>

<https://www.tutorialspoint.com/uml/>

## Types of UML

System development focuses on three overall different system models

* Functional: there are use case diagrams, which describe system functionality from the point of the user
* Object: these are class diagrams, which describe the structure of the system in terms of objects, attributes, associations, and operations.
* Dynamic: Interaction diagrams, state machine diagrams, and activity diagrams are used to describe the internal behaviour of the system.

These system models are visualised through two different types of diagrams: structural and behavioural.

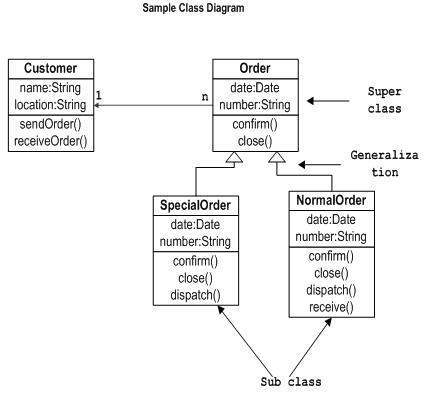
### Object Orientated Concepts in UML

* Objects: represent an entity and the basic building block
* Class: blue print of an object
* Abstraction: Behaviour of a real-world entity
* Encapsulation: mechanism of binding data together and hiding them from the outside world
* Inheritance: mechanism of making new classes from existing ones
* Polymorphism: it defines the mechanism to exist in different forms

### Structural Diagrams

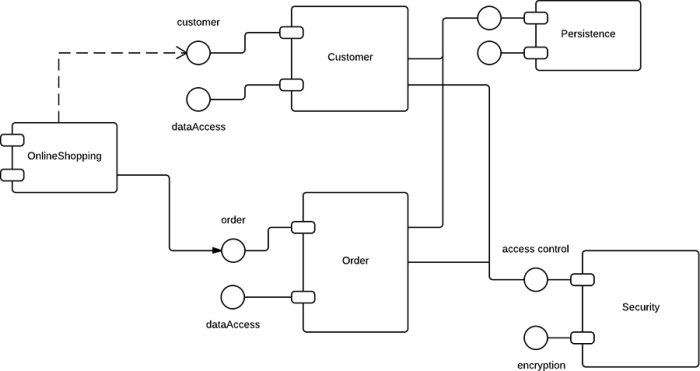
* **Class Diagram**: The most commonly used UML diagram, and the principal foundation of any object orientated solution. It represents the static view of an application. Can be used for constructing executable code of a software application.

1. Classes are represented with a rectangular shape that is split into thirds. The top section is the class name, the middle contains class attributes and the bottom section features the class operations (methods).
2. Lines and arrows between classes represent relationships between them.

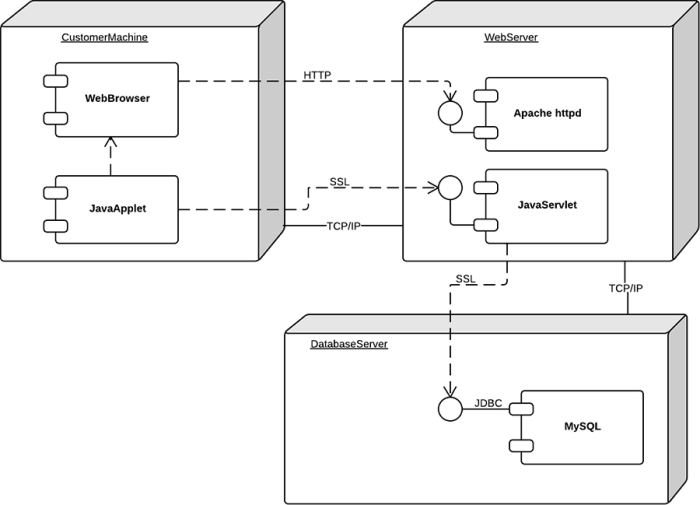


* **Component Diagram:** Displays the structural relationships of software systems elements, most often employed when working with complex systems with complex components. Components communicate using interfaces.

1. Represent a component with a rectangle shape. It should have two small rectangles on the side, or feature an icon with this shape.
2. Add lines between component shapes to represent the relevant relationships.



* **Composite Structure Diagram:** Used to show the internal structure of a class.
* **Deployment Diagram:** Illustrates system hardware and its software. Useful when software is deployed across multiple machines with unique configurations.

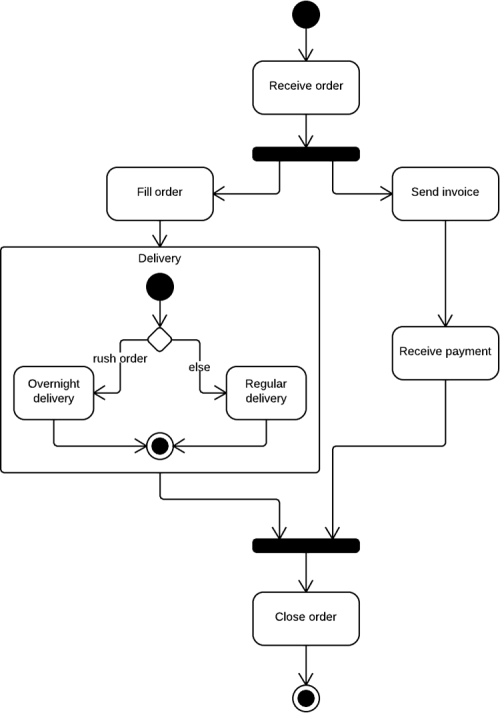
1. When drawing a deployment diagram, use the same notation that you use for a component diagram.
2. Use a 3-D cube to model a node (which represents a physical machine or virtual machine).
3. Label the node in the same style that is used for sequence diagrams. Add other nodes as needed, then connect with lines.

* **Object Diagram:** Shows the relationship between objects using real world examples and illustrate how a system will look at any given time. Because data is available within objects, they can be used to clarify relationships between objects.
* **Package Diagram:** There are two special types of dependencies defined between packages: package import and package merge. Packages can represent the different levels of a system to reveal the architecture. Package dependencies can be marked to show the communication mechanism between levels.

### Behavioural Diagrams

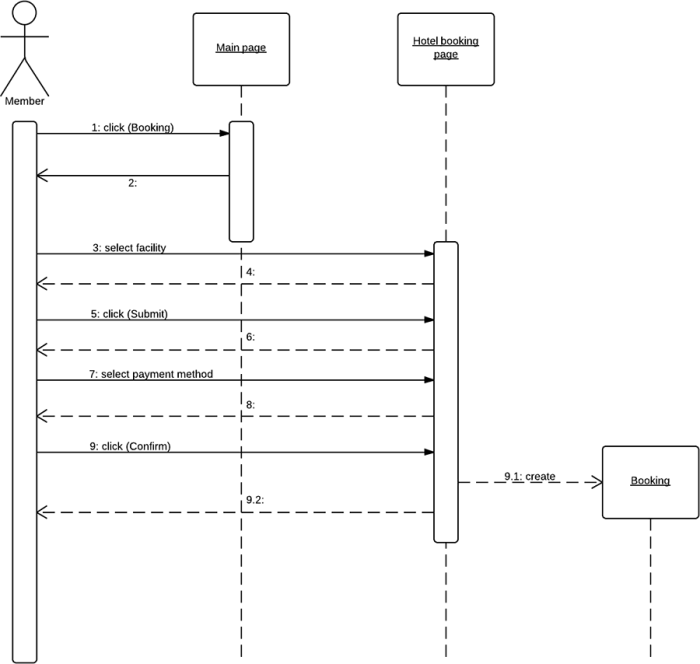
* **Activity Diagram:** graphically represented business or operational workflows to show the activity of any part or component in the system. Used as an alternative to State Machine Diagrams

1. Begin your activity diagram with a solid circle.
2. Connect the circle to the first activity, which is modeled with a round-edged rectangle.
3. Now, connect each activity to other activities with lines that demonstrate the stepwise flow of the entire process.



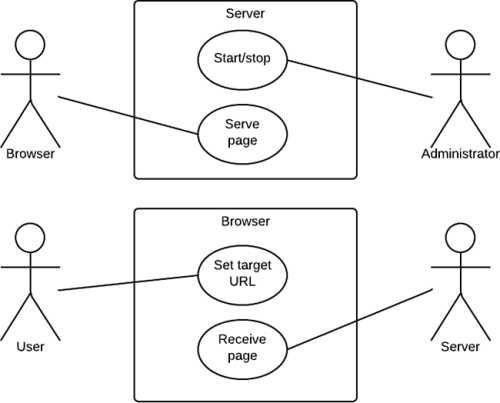
* **Communication Diagram:** Similar to sequence diagrams, but the focus is on messages passed between objects. The same information can be represented using a sequence diagram and different objects.
* **Interaction Overview Diagram:** There are 7 types of interaction diagram. This diagram shows the sequence in which they act.
* **Sequence Diagram:** Shows how objects interact with each other and the order of occurrence. Represent interactions for a scenario. Illustrate how processes interact with each other by showing calls between different objects in a sequence. These diagrams have two dimensions: vertical and horizontal. The vertical lines show the sequence of messages and calls in chronological order, and the horizontal elements show object instances where the messages are relayed.

1. To create a sequence diagram, write the class instance name and class name in a rectangular box.
2. Draw lines between class instances to represent the sender and receiver of messages.
3. Use solid arrowheads to symbolize synchronous messages, open arrowheads for asynchronous messages, and dashed lines for reply messages



* **State machine Diagram:** Similar to activity diagrams, they describe the behaviour of objects that behave in varying ways in their current state.
* **Timing Diagram:** The behaviour of objects in a given time frame are represented. If there is a single object, the diagram is simple. With more than one object, interactions of objects are shown during that particular time frame.
* **Use Case Diagram:** Represents a particular functionality of a system, created to illustrate how functionalities relate and their internal/external controllers (actor). A use case is a list of steps that define interaction between an actor (a human who interacts with the system or an external system) and the system itself. Use case diagrams depict the specifications of a use case and model the functional units of a system. These diagrams help development teams understand the requirements of their system, including the role of human interaction therein and the differences between various use cases. A use case diagram might display all use cases of the system, or just one group of use cases with similar functionality.

1. To begin a use case diagram, add an oval shape to the centre of the drawing.
2. Type the name of the use case inside the oval.
3. Represent actors with a stick figure near the diagram, then use lines to model relationships between actors and use cases.

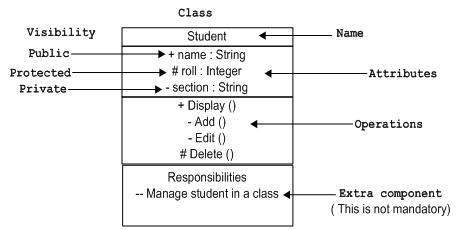


## Basic Notation of UML

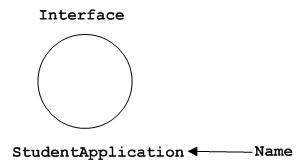
UML notations are the most important elements in modelling. Efficient and appropriate use of notations is very important for making a complete and meaningful model. The model is useless, unless its purpose is depicted properly. UML diagrams are made using the notation of things and relationships. Extensibility is another thing which makes UML more powerful and flexible.

### Structural Things

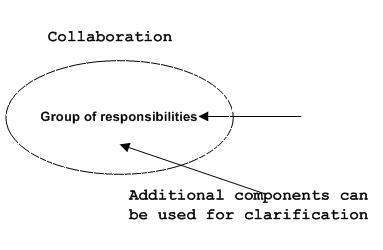
* **Class:** Classes are used to represent objects. Objects can be anything having properties and responsibility.



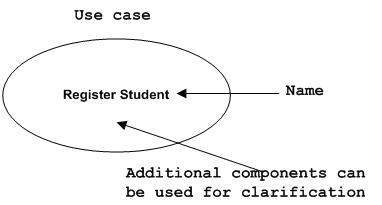
* **Object Notation**: Represented in that same way as a class. The only difference is that the name is underlined. The object is an actual implementation of a class, which is known as the instance of a class. It has the same usage as the class.
* **Interface Notation**: Represented by a circle. The name is generally written below the circle but it can be found above in some diagrams. Interface is used to describe the functionality without implementation.



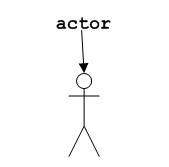
* **Collaboration Notation:** represented by a dotted eclipse. It has a name written inside the eclipse. Collaboration represents responsibilities, generally responsibilities are in a group.



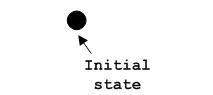
* **Use Case Notation:** Represented as an eclipse with a name inside it. It may contain additional responsibilities. Used to capture high level functionalities of a system.



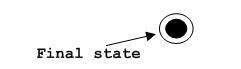
* **Actor Notation:** An actor can be defined as some internal or external entity that interacts with the system.



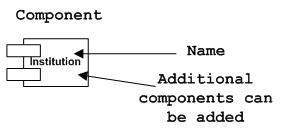
* **Initial State Notation:** Is defined to show the start of a process.



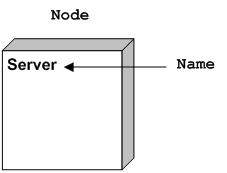
* **Final State Notation:** Is defined to show the end of a process.



* **Active Class Notation:** Similar to a class with a solid border. (The border is thicker). Used to describe the concurrent behaviour of a system.
* **Component Notation:** Component is used to represent any part of a system for which UML diagrams are made.



* **Node Notation:** A node represent the physical component of the system such as the server, network, etc.

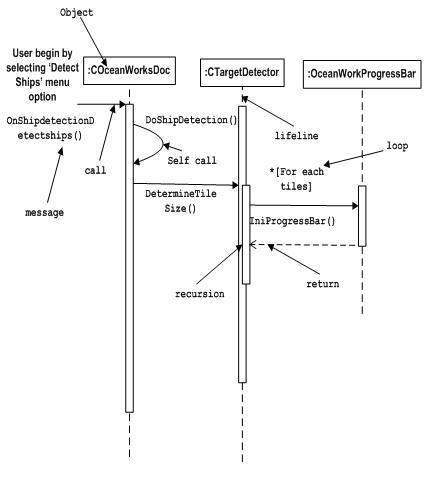


### Behavioural Things

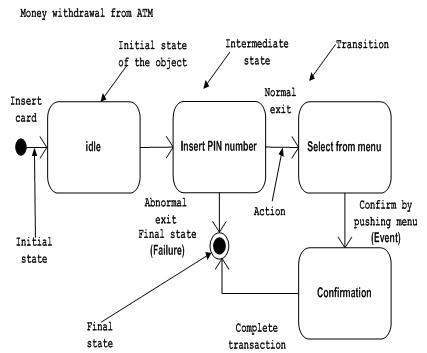
Dynamic parts are one of the most important elements in UML. These features include interactions and state machines.

Interactions can be of two types:

* Sequential (sequence diagram)
* Collaborative (collaboration diagram)
* **Interaction Notation:** Interaction is basically a message exchange between two UML components. Interaction is used to represent the communication among components of a system.

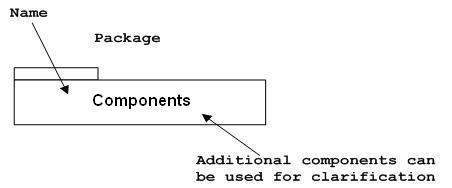


* **State Machine Notation:** State machine describes the states of a component in its lifecycle. The state can be idle, active, or any other depending on the situation.



#### Grouping Things

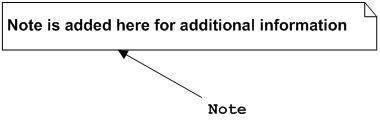
* **Package Notation:** Used to wrap the components of the system.



#### Annotational Things

Explanation of different elements and their functionalities are very important. UML has notes notation to support this requirement.

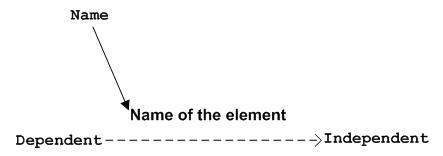
* **Note Notation:**  Used to provide necessary information of a system.



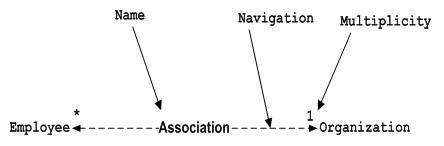
#### Relationships

A model is not complete unless relationships between elements are described properly. The Relationship gives proper meaning to a UML model.

* **Dependency Notation**: It describes the dependent elements and the direction of the dependency.



* **Association Notation:** Describes how the elements in a UML diagram are associated by describing how many elements are taking part in an interaction. Multiplicity is also mentioned (0-none, 1-one, \*-many) to show how many objects are associated.



* **Generalisation Notation:** Generalisation describes the inheritance relationship of the object orientated world. It is a parent and child relationship.

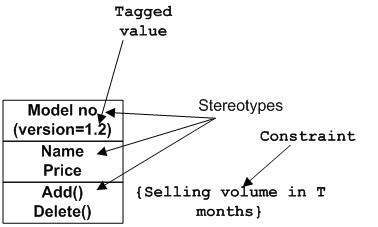


* **Realisation Notation:** A relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This exists in cases of interfaces.

Realization

* **Extensibility Notation:** Used to enhance the power of the language. It is additional elements used to represent some extra behaviour of the system that are not covered by the standard available notations.

1. Stereotypes (Represents new elements)
2. Tagged values (Represents new attributes)
3. Constraints (Represents the boundaries)



# More on Use Case Diagrams

A use case is a description of a system’s behaviour from a user’s standpoint. It describes how the user uses a system to accomplish a particular goal.

A use case is a (usually high-level) user activity in the system. It is a collection of related success and failure scenarios that describe actors using a system to support a goal. The use case name should begin with a verb.

Use case diagrams consist of at least 2 components:

* **An Actor:** represented as stick people with a label naming the actors role.
* **A Use Case:** represented as eclipses with a label inside naming the use case

An actor represents a role that a user might play. Each role must be represented separately.

Actor and use case names must be unique within a given diagram.

Consider use cases to be similar to classes:

* A use case describes an activity that is possible
* A given system may have several instances of that activity throughout its lifetime
* A use case is a unit of behaviour, and a class a unit of software.

# More on Class Diagrams

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

#### Components of a Class Diagram

* Class name
* Attributes
* Operations (methods)
* Multiplicity (cardinality)
* 1: no more than one
* 0.1: zero or one
* \*: many
* 0..\*: zero or many
* 1..\*: one or many
* Association (a set of links connecting elements of a UML model).

# More on Sequence Diagrams

Are used primarily to show the interactions between objects in the sequential order in which the occur.

Technical staff can find sequence diagrams useful in documenting how a future system will behave.

* **Object:** this class represents a class or object. They demonstrate how an object will behave in context of the system. Class attributes should not be listed in this shape.
* **Activation bar:** symbolised by a rectangle shape, represents the time needed for an object to complete a task. The longer a task will take the longer a box will become.
* **Actors:** a stick figure. Entities that are both interactive and external to the system.
* **Lifeline:** a dashed vertical line that represents the passage of sequential events that occur during the chartered process.
* **Messages:** packets of information that are transmitted between objects. They may reflect the start and execution of an operation, or the sending and reception of a signal.

# Database Concepts

## What is a File Processing System?

A file processing system is a method for storing and organising computer files and the data they contain to make it easy to find and access them. The may use a storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files.

## Characteristics of the File Processing System

* A group of files storing data of an organisation/business.
* Each file is independent of one another.
* Each file is called a flat file.
* Each file is contained and processes information for one specific purpose.
* Physical implementation and access are written into database application.
* As systems become more complex, file processing systems offer little flexibility, present many limitations, and are difficult to maintain

## Advantages

* Easy to use
* Keeps files organised for a business either alphabetically or numerically

## Disadvantages

* **Duplication of data:** same information is stored in more than one file. It is wasteful, it costs and money to enter data more than once. It takes up additional storage space leading to additional costs. Can lead to loss of data integrity so data is no longer consistent.
* **Data dependence:** files and records that are described by specific physical formats are coded into an application program by a programmer. If the format of a certain record is changed, the code in each file containing that format must be updated. Instructions for data storage and access are written into the applications code. Changes in storage structure or access methods could greatly affect the processing or results of an application.
* **Difficulty in presenting data from the user’s view:** to create useful applications for the user, often data from various files are combined. In file processing, it was difficult to determine relationships between isolated data to meet the user requirements.
* **Data inflexibility:** program data interdependency and data isolation, limited the flexibility of life of file processing systems in providing users with ad-hoc information requests.
* **Incompatible file formats:** as the structure of files is embedded in the application programs, the structures are dependent on the application programming language.
* **Data security:** the security of data is low in file based systems because, the data is maintained in the flat file(s) is easily accessible.

# What is A Database?

A database is a collection of data or information which is held together in a logical or organised way. They are designed to offer an organised mechanism for storing, managing and retrieving information.

## Characteristics of a Database

* A command language that allows you to create, alter and delete the database
* A way of documenting all the internal structures that make up the database (data dictionary)
* A language to support the manipulation and processing of data
* Support to view the database from different viewpoints according to the requirements of the user
* Provide some level of security and access control to the data

## Advantages Of Databases

* **Reduced data redundancy:** data is stored only once
* **Improved data integrity:** data is stored only once for each entity, don’t need to update multiple records for the same entity
* **Easier updating of data**
* **Data and program independence:** the data files are separate from the applications and thus can be used by many applications
* **Improved strategic use of data:** backups and access can be better controlled by one database than multiple files of data.

## Disadvantages of Databases

* **More complex:** it must support many applications, agreement of terms
* **More difficult to recover from a failure:** failure to a system effects all applications since all share the data
* **More expensive:** need specialised professionals

## Advantages of Database Approach to Traditional File Approach

* It allows the same number of users to access the same pieces of data at the same time
* It allows files to be edited and changed while traditional file processing does not
* It allows certain people or users such as administrators to have more control than other users, whereas in file processing, all users have the same amount of control
* Reduced data redundancy: data is stored only once in a database while in traditional file approach data may have been duplicated
* Can be accessed remotely with username and password