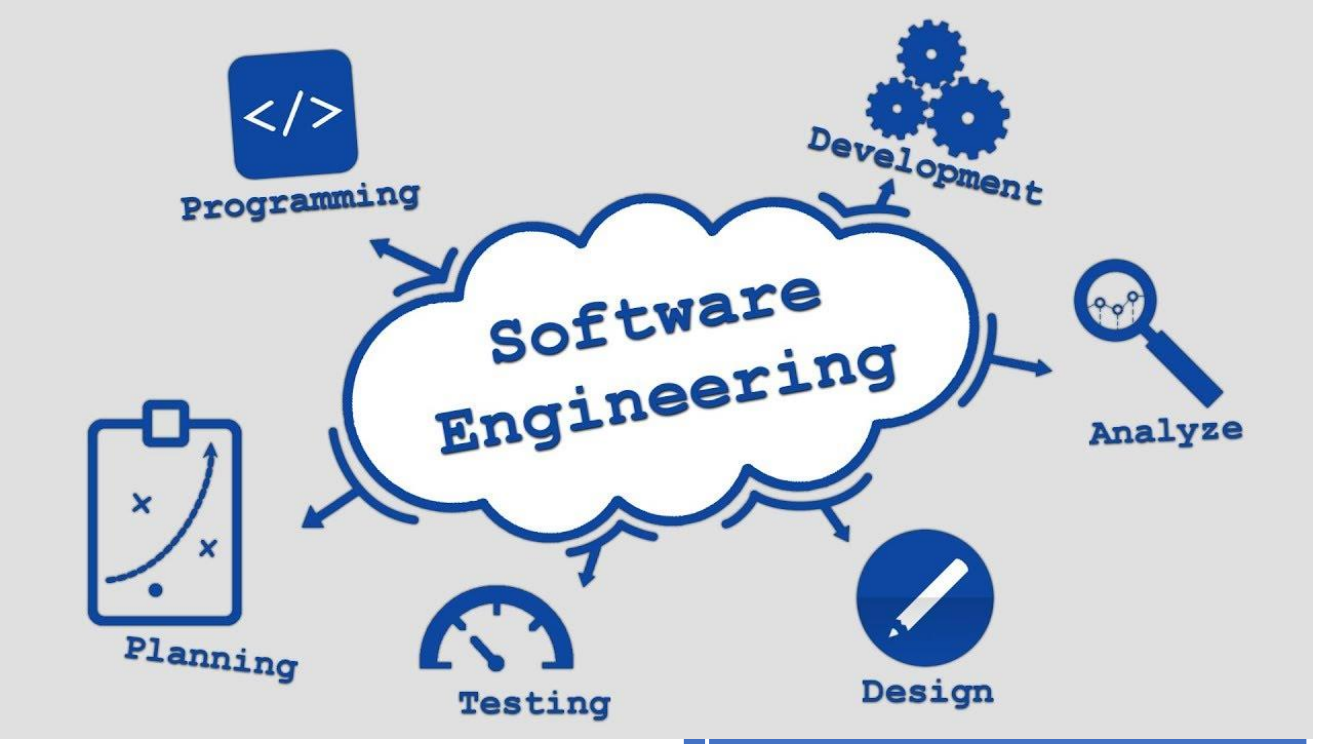
E-Book

Software Engineering - DSE



DSE

National Institute of Business Management

E-Book

Table of Contents

List of Figures

List of Tables

Lesson 01 – Software Application Domains

What is a Software?

Software is a collection of computer programs, procedures, rules and associated documentation and data pertaining to the operation of a computer system.

Software is traditionally divided into two categories.

- **System Software**
- **Application Software**

Figure 1.0.1 Software vs Hardware

System Software: Designed to operate and control the computer hardware and provide a platform for running the application software. Operating Systems and computer drivers.

Figure 1.0.2 Examples for System Software

Application Software: Stand-alone programs that solve a specific business need. Office package, Skype etc.

Figure 1.0.3 Examples for Application Software

Engineering/Scientific Software: Characterized by number crunching algorithms such as automotive stress analysis, space engineering.

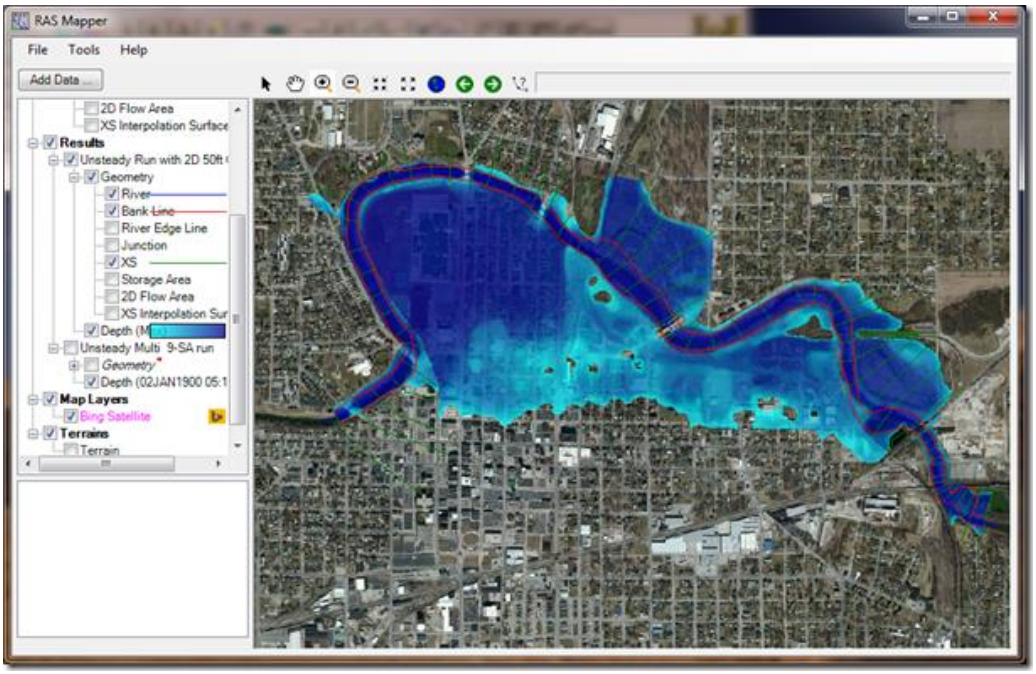


Figure 1.0.4 Engineering/Scientific software

Embedded Software: Resides within a product or system and is used to implement and control features and functions for the end user and for the system itself.

Ex: Key pad control for microwave oven, Anti-lock brakes, Auto-focus camera, Teller machines.

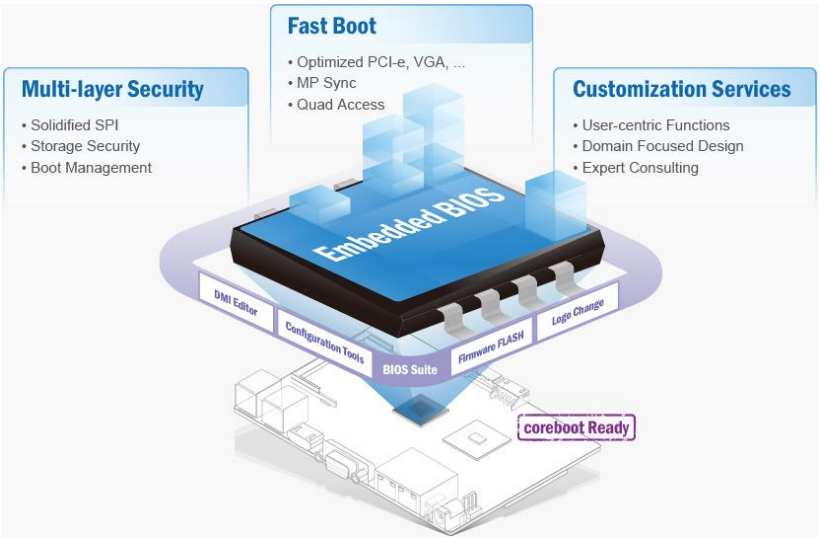


Figure 1.0.5 Embedded Software

Product Line Software: Focus on a limited marketplace to address mass consumer market.

Ex: Database management systems, Adobe Photoshop

Figure 1.0.6 Product Line Software

Web Applications: The network-centric software category spans a wide array of applications.

Figure 1.0.7 Web applications

Open Source: Free source and free tools open to the computing community.

Figure 1.0.7 Open Source Software

Software Engineering & Software Engineer

What is Software Engineering?

The application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

Software Engineering is a Layered Technology.

Figure 1.0.8 Software Engineering Layers

- **Quality Focus**: The bedrock that support SE is quality focus.

- **Process**: The foundation of the SE. Process defines a framework that must be established for effective delivery of software engineering technology.

- **Methods**: Provide the technical for building software. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, testing and support.

- **Tools**: Provide automated or semi-automated support for the process and the methods.

Importance of Software Engineering

The economies of all developed nations are dependent on software.

Individuals, business and governments increasingly rely on software for strategic and tactical decision making as well as day to day operations and control.

If the software fails it will be a huge impact to those parties. It follows that software should exhibit high quality.

Attributes of a Good Software

- **Maintainability**

  The software should be written in a way that it can be evolve to meet changing needs of the customer.

- **Dependability**

  A software must be Trustworthy, Reliable, Safe and Secured.

- **Efficiency**

  A software should be efficient in every way. The software should not make wasteful of system resources (ex. memory, processing cycles).

- **Acceptability (Usability)**

  The software must be acceptable to the group of users for which it's designed for.

Key Challenges of a Software Engineer

- **The Legacy Challenge**

  The legacy challenge is the challenge of maintaining and updating old Software.

- **The Heterogeneity Challenge**

  The heterogeneity challenge increasingly, systems are required to operate as distributed systems across networks that include different types of computers and with different kinds of support systems.

- **The Delivery Challenge**

  The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.

- **Update the knowledge & qualifications to latest technologies**

  It is a challenge to update the knowledge and qualifications while working in the industry.

- **Technical Challenges**

It is a challenge to solve day to day technical issues of systems without any support.

- **Dealing with Management & Customer**

  Dealing with your Manager and Customer is the most difficult challenge the Software Engineer ever face.

- **Operational Challenges**

  It is a challenge to run the day to day operations in the company.

Ethical Responsibilities of Software Engineer

Software engineers must behave in an ethical and morally responsible way if they are to be respected as professionals.

- **Confidentiality**

  Engineers should normally respect the confidentiality of their employers or clients.

- **Competence**

  Engineers should not misrepresent their level of competence.

- **Intellectual Property Rights**

  Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc.

- **Computer Misuse**

  Software engineers should not use their technical skills to misuse other people's computers.

Seven Core Principles of Software Engineering

1) **The First Principle: The Reason It All Exists:**

   A software system exists for one reason: to provide value to its users. Ask yourself questions such as: "Does this add real value to the system?"

2) **The Second Principle: KISS (Keep It Simple, Stupid!):**

   All design should be as simple as possible

3) **The Third Principle: Maintain the Vision:**

A clear vision is essential to the success of a software project.

4) **The Fourth Principle: What You Produce, Others Will Consume:**

In some way or other, someone else will use, maintain, document, or otherwise depend on being able to understand your system.

5) **The Fifth Principle: Be Open to the Future:**

In today software lifetimes are typically measured in months instead of years because of the business environment change.

6) **The Sixth Principle: Plan Ahead for Reuse:**

Reuse saves time and effort. Achieving a high level of reuse is arguably the hardest goal to accomplish in developing a software system.

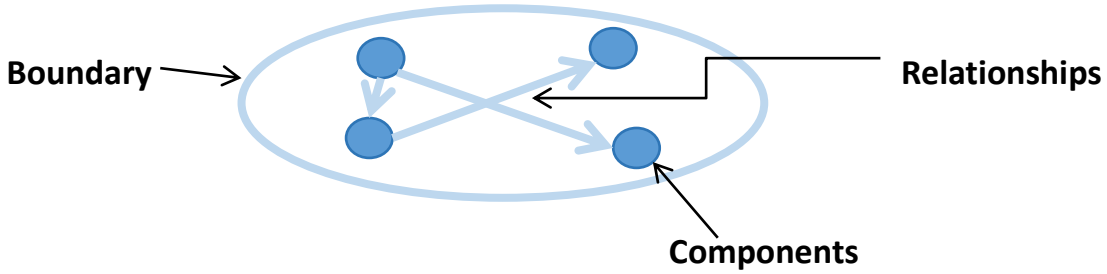
7) **The Seventh principle: Think:**

This last principle is probably the most overlooked. Placing clear, complete thought before action almost always produces better results.

Lesson 02 – Software Development Life Cycle

What is a System?

A collection of interrelated parts which act as a whole towards a common goal.



Examples:

- Nervous System
- Communication System
- Business

  (Components of a business System - Marketing, Sales, Manufacturing, Accounts etc.)

Types of Systems

- **Physical**

  Physical system is tangible entities that may be static or dynamic in nature.

- **Open and Closed**

  An open system continually interacts with its environment. A closed system is isolated from environment influences.

- **Sub System and Super System**

  Each system is part of a large system. Sub systems are the smaller systems within a system. Super system denotes extremely large and complex system.

- **Permanent and Temporary System**

  A permanent system is a system enduring for a time span that is long relative to the operation of human. Temporary system is one having a short time span.

- **Natural and Man Made System**

  System which is made by man is called man made system. Systems which are in the environment made by nature are called natural system.

- **Deterministic and Probabilistic**

  A Deterministic system is one in which the occurrence of all events is perfectly predictable. Probabilistic system is one in which the occurrence of events cannot be perfectly predicted.

- **Man-made Information System**

  An information system is the basis for interaction between the user and the analyst. It determines the nature of relationship among decision makers.

- **Natural Systems**

  These are systems that occur in nature, and together they form the ecosystems that make up our natural environment.

Eg: Circulation of Water in our ocean, Weather and Climate Systems, Energy Cycles, Food Chain.

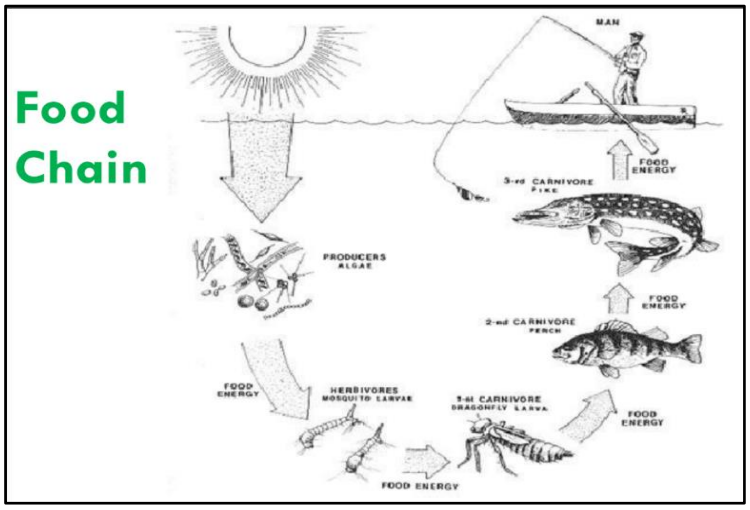


Figure 2.0.1 Natural Systems

- **Information Systems**

  These are systems that are created by people to fill individual and collective needs and wants.

Eg: Human settlements, Transportation routes, Communication systems, Economics, Infrastructure, Energy

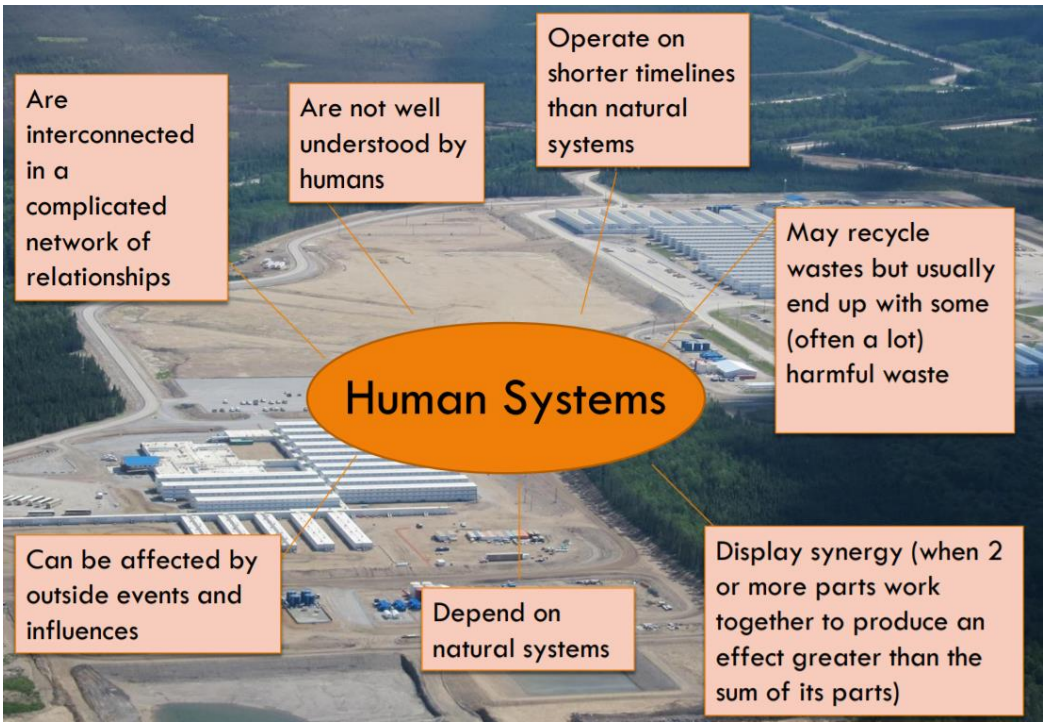


Figure 2.0.2 Human Systems

- **Open Systems**

  Open systems refer to systems that interact with other systems or the outside environment. The boundaries of open systems, because they interact with other systems or environments, are more flexible than those of closed systems, which are rigid, and largely impenetrable.

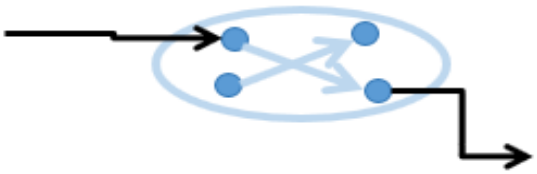


Figure 2.0.3 Open Systems

- **Closed Systems**

  Closed systems refer to systems having relatively little interaction with other systems or the outside environment. A closed system perspective views organizations as relatively independent of environmental influences.

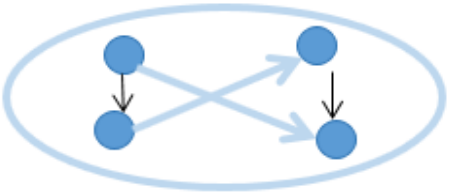


Figure 2.0.4 Closed Systems

What is an Information System?

It is an integrated set of components for collecting, storing, processing, and communicating information.

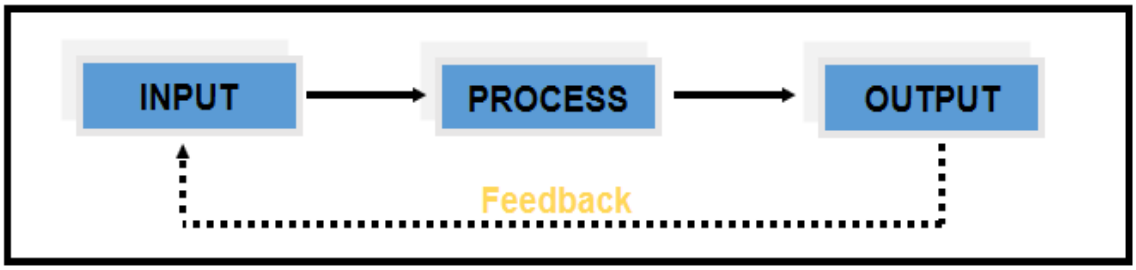


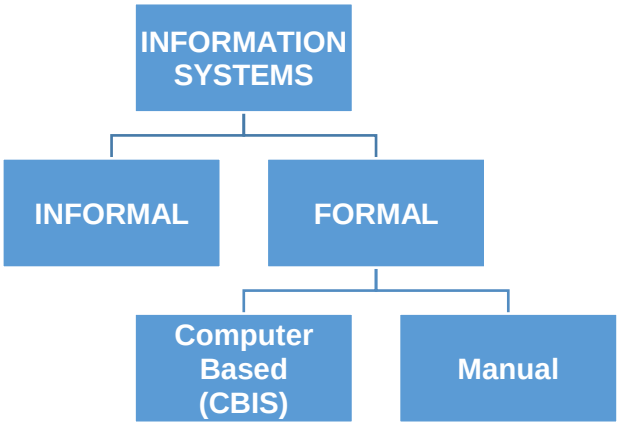
Figure 2.0.5 Flow Diagram of Information System

Components of a Computerized Information System

- Hardware
- Software
- Data Bases
- Personnel

Why Information Systems?

Business firms, other organizations, and individuals in contemporary society rely on information systems to manage their operations, compete in the marketplace, supply services, and augment personal lives.



Types of Information Systems

- Office Automation Systems (OAS)
- Transaction Processing Systems (TPS)
- Management Information Systems (MIS)
- Decisions Support Systems (DSS)
- Executive Support Systems (ESS)
- Geographical information systems (GIS)
- Knowledge Management Systems (KMS)
- Content Management Systems (CMS)
- Enterprise Resource Planning Systems (ERP)
- Smart systems

Office Automation Systems:

The office automation systems**,** also called OAS consist of applications designed to help the daily work of the administration of an organization, are part of this type of software the word processors, the leaves calculation, the editors of presentations, customer email, etc. When several of these applications are grouped into a single software package for easy distribution and installation, the set is known by the name of office suite.

- Perhaps the most popular software package that can fit the definition of OAS (and to the office suite) is Microsoft Office in all its versions.
- Raw data storage, electronic transfer, and the management of electronic business information comprise the basic activities of an office automation system. Office automation helps in optimizing or automating existing office procedures.

Advantages are:

- Office automation can get many tasks accomplished faster.
- It eliminates the need for a large staff.
- Less storage is required to store data.
- Multiple people can update data simultaneously in the event of changes in schedule.

Transaction Processing System:

A Transaction Processing System (TPS) is a type of information system that collects, stores, modifies and retrieves the data transactions of an enterprise. The success of commercial enterprises depends on the reliable processing of transactions to ensure that customer orders are met on time. The field of transaction processing, therefore, has become a vital part of effective business management.

Examples for TPS:

- Hotel Reservation Systems
- Payroll systems
- Order Entry Systems
- Airline ticket reservation system

Features of Transaction Processing Systems

- Rapid response – fast performance with rapid results.
- Reliability – well-designed backup and recovery with a low failure rate.
- Inflexibility – treat every transaction equally.
- Controlled processing – maintain specific requirements for the roles and responsibilities of different employees.

Types of TPS

- **Batch Processing system**: Batch processing is where the information is collected as a batch and then processed later on. An example of batch processing is paying by cheque. Batch processing is useful for enterprises that need to process large amounts of data using limited resources.
- **Real Time Processing**: Real time processing is where all details of the transaction are recorded and changed at the time as it occurs.

  Examples of real time processing are ATM's.

Management Information Systems:

A management information system (MIS) is an information system used for decision-making, and for the coordination, control, analysis, and visualization of information in an organization.

It produces regular reports on operations for every level of management in a company. The main purpose of the MIS is to give managers feedback about their own performance; top management can monitor the company as a whole.

MIS extract, process and summarizes data from the TPS and provides periodic reports.

Decision Support System:

A decision support system (DSS) is a computer-based application that collects, organizes and analyzes business data to facilitate quality business decision-making for management, operations and planning. DSS analysis helps companies to identify and solve problems, and make decisions.

- DSSs include knowledge-based systems. A properly designed DSS is an interactive software-based system intended to help decision makers compile useful information from a combination of raw data, documents, and personal knowledge, or business models to identify and solve problems and make decisions.
- A decision support system may present information graphically and may include an expert system or artificial intelligence (AI).

Typical information that a decision support application might gather and present would be:

- Comparative sales figures between one week and the next.
- Projected revenue figures based on new product sales assumptions.
- The consequences of different decision alternatives, given past experience in a context that is described.

Executive Support Systems:

Executive Support System (ESS) is a reporting tool (software) that allows you to turn your organization's data into useful summarized reports. These reports are generally used by executive level managers for quick access to reports coming from all company levels and departments such as billing, cost accounting, staffing, scheduling, and more.

In addition to providing quick access to organized data from departments, some Executive Support System tools also provide analysis tools that predicts a series of performance outcomes over time using the input data.

Advantages of EIS

- Easy for upper-level executives to use, extensive computer experience is not required in operations.
- Provides strong drill-down capabilities to better analyze the given information.
- Information that is provided is better understood.
- EIS provides timely delivery of information. Management can make decisions promptly.
- Improves tracking information.
- Offers efficiency to decision makers.

Geographical Information Systems:

A geographic information system (GIS) is a system designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data. GIS applications are tools that allow users to create interactive queries (user-created searches), analyze spatial information, edit data in maps, and present the results of all these operations.

Geographic Information Systems really comes down to just 4 simple ideas: These are the primordial functions of a GIS.

- Create geographic data
- Manage it
- Analyze it and
- Display it on a map

GIS can be used to Mapping where things are, Mapping quantities, Mapping densities, finding what is inside, finding what is nearby, Mapping changes.

Following are some open-source desktop GIS projects.

- GRASS GIS – Originally developed by the U.S. Army Corps of Engineers: a complete GIS.

- ILWIS (Integrated Land and Water Information System) – Integrates image, vector and thematic data.

- MapWindow GIS – Free desktop application and programming component.

- QGIS (previously known as Quantum GIS) – Runs on Linux, Unix, Mac OS X and Windows.

- SAGA GIS (System for Automated Geoscientific Analysis) –- A hybrid GIS software. Has a unique Application Programming Interface (API) and a fast-growing set of geoscientific methods, bundled in exchangeable Module Libraries.

Knowledge Management Systems

Knowledge management (KM) is the process of creating, sharing, using and managing the knowledge and information of an organization. Knowledge management efforts typically focus on organizational objectives such as improved performance, competitive advantage, innovation, the sharing of lessons learned, integration and continuous improvement of the organization.

A knowledge base software allows the company to publish necessary documents for organization-wide consumption. With right knowledge management tools, the company can put together a collection of best practices, tips for customer support and other documents for workforce enablement.

Best Knowledge Management Software:

- **LiveAgent** is the Most reviewed and #1 Rated help desk software for SMB. Companies like BMW, Yamaha, Huawei, Orange use LiveAgent to deliver customer wow to 150M end users worldwide.

- **Zoho** Connect is a team collaboration app, that unifies people, resources, and the apps they need. Users can share ideas, hold real-time discussions, contact anyone in the network, create their own apps, build their knowledge base.

- **Bitrix24** is a leading free social knowledge management and collaboration platform used by over 4 million companies worldwide. Available in cloud and on-premise with open source code access. Share and discuss ideas, manage knowledge, manage projects and do more with Bitrix24.

- **Confluence** is an open and shared workspace that connects people to the ideas and information they need to do their best work.

Content Management System

A content management system (CMS) is a software application or set of related programs that are used to create and manage digital content.

Features of CMS

- Content creation (allows users to easily create and format content)
- Workflow management (assigns privileges and responsibilities based on roles such as authors, editors and admins)
- Intuitive indexing, search and retrieval features index all data for easy access through search functions and allow users to search by attributes such as publication dates, keywords or author.
- Revision features allow content to be updated and edited after initial publication. Revision control also tracks any changes made to files by individuals.
- Publishing functionality allows individuals to use a template or a set of templates approved by the organization, as well as wizards and other tools to create or modify content.

One major advantage of a CMS is its collaborative nature. Multiple users can log on and contribute, schedule or edit content to be published. Because the interface is usually browser-based, a CMS can be accessed from anywhere by any number of users.

The second major advantage of a CMS is that it allows non-technical people who don't know programming languages to easily create and manage their own web content.

CMS Examples

- Joomla
- Magento
- ModX
- Wix
- Weebly
- Wordpress

Enterprise Resource Planning Systems

Enterprise resource planning (ERP) is business process management software that allows an organization to use a system of integrated applications to manage the business and automate many back office functions related to technology, services and human resources.

ERP software typically integrates all facets of an operation including product planning, development, manufacturing, sales and marketing in a single database, application and user interface.

Some of the most common ERP modules include those for product planning, material purchasing, inventory control, distribution, accounting, marketing, finance and HR.

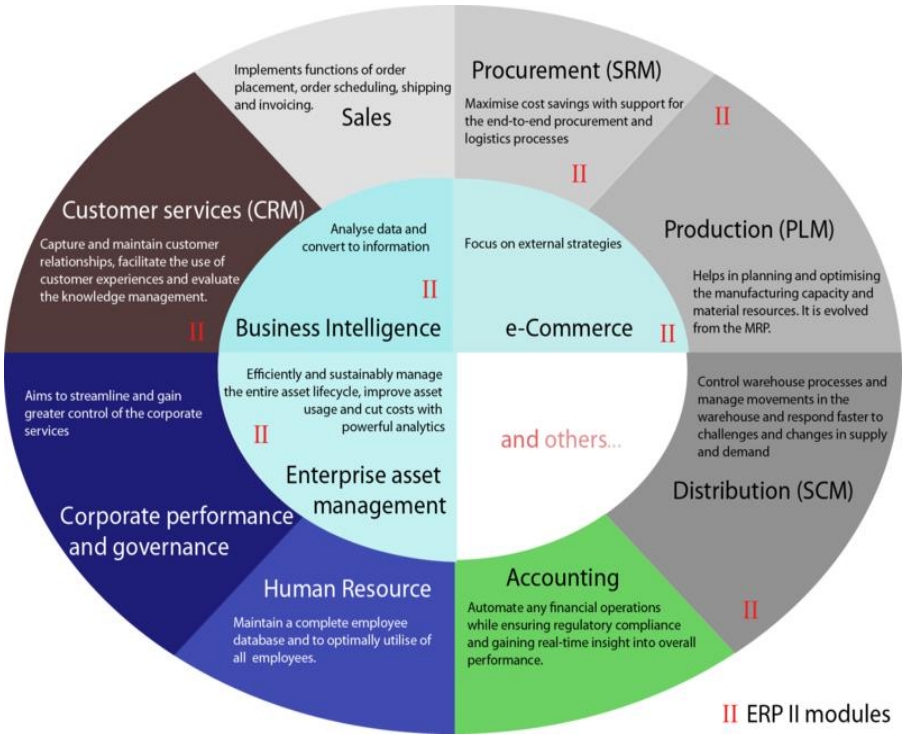


Figure 2.0.6 ERP System

The following new and continuing computing trends have an impact on the growth of enterprise ERP software:

- **Mobile ERP:** Executives and employees want real-time access to information, regardless of where they are. It is expected that businesses will embrace mobile ERP for the reports, dashboards and to conduct key business processes.
- **Cloud ERP:** The cloud has been advancing steadily into the enterprise for some time, but many ERP users have been reluctant to place data in the cloud. Those reservations have gradually been evaporating, however, as the advantages of the cloud become apparent.
- **Social ERP:** There has been much hype around social media and how important —or not — it is to add to ERP systems. Certainly, vendors have been quick to seize the initiative, adding social media packages to their ERP systems with much fanfare. But some wonder if there is really much gain to be had by integrating social media with ERP.

Smart Systems

Smart systems incorporate functions of sensing, actuation, and control in order to describe and analyze a situation, and make decisions based on the available data in a predictive or adaptive manner, thereby performing smart actions. In most cases the "smartness" of the system can be attributed to autonomous operation based on closed loop control, energy efficiency, and networking capabilities.

- **First-generation smart systems**: object recognition devices, driver status monitoring, and multifunctional devices for minimally invasive surgery.
- **Second-generation smart systems**: active miniaturized artificial organs like cochlear implants or artificial pancreas, advanced energy management systems, and environmental sensor networks.
- **Third-generation smart systems**: combine technical "intelligence" and cognitive functions so that they can provide an interface between the virtual and the physical world.

Software Development Life Cycle (SDLC)

- SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software.
- The life cycle defines a methodology for improving the quality of software and the overall development process.
- Traditionally, the Software Development Life Cycle consisted of five stages. That has now increased to seven phases. Increasing the number of steps helped systems analysts to define clearer actions to achieve specific goals.

Seven phases of the SDLC

1. Planning
2. Systems Analysis and Requirement Gathering
3. Systems Design
4. Development (Coding)
5. Integration and Testing
6. Implementation (Deployment)
7. Operations and Maintenance

1. Planning

It identifies whether or not there is the need for a new system to achieve a business's strategic objectives. This is a **preliminary plan (or a feasibility study)** for a company's business initiative to acquire the resources to build on an infrastructure to modify or improve a service.

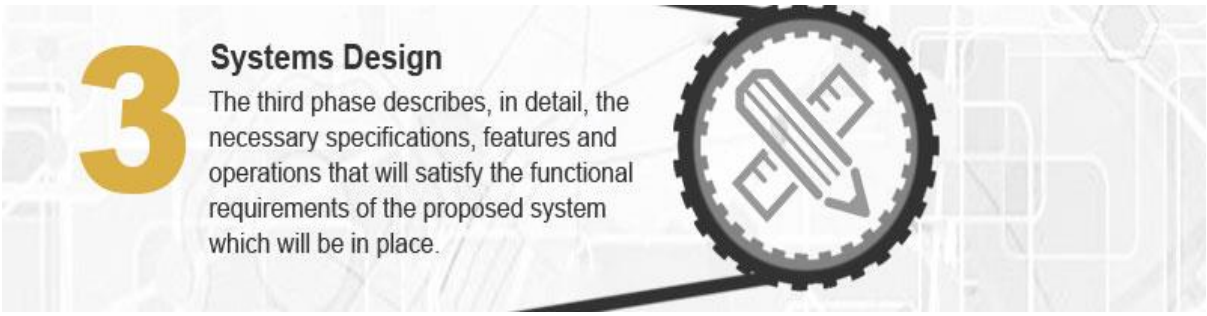
The purpose of this step is to find out the **scope of the problem and determine solutions**. Resources, costs, time, benefits and other items should be considered at this stage.

2. Systems Analysis and Requirement Gathering

The second phase is where businesses will work on the source of their problem or the need for a change. In the event of a problem, possible solutions are submitted and analyzed to identify the best fit for the ultimate goal(s) of the project. **This is where teams consider the functional requirements of the project or solution.** It is also where system analysis takes place—or analyzing the needs of the end users to ensure the new system can meet their expectations.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created.

3. Systems Design



The third phase describes, in detail, the necessary specifications, features and operations that will satisfy the functional requirements of the proposed system which will be in place. It brings down whole knowledge of requirements and analysis on the desk and design the software product. The output of this step comes in the form of two designs; logical design and physical design. **Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.**

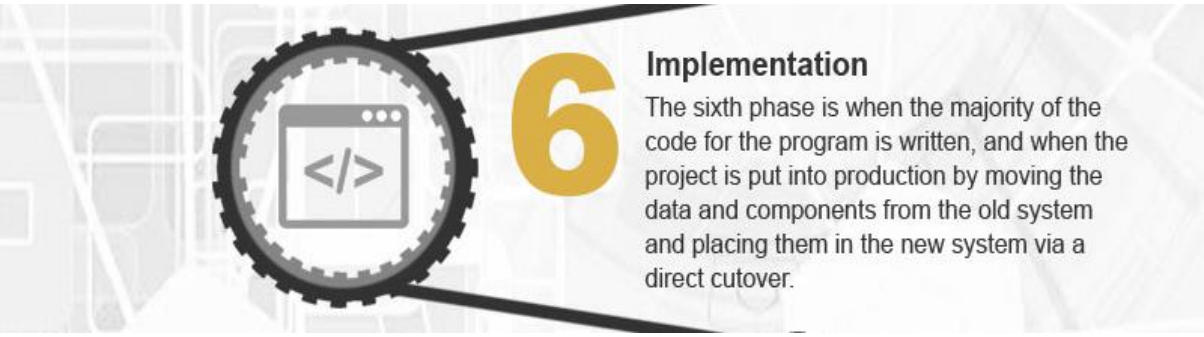
4. Development

The fourth phase is **when the real work begins**—in particular, when a programmer, network engineer and/or database developer are brought on to do the major work on the project. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

5. Integration and Testing

The fifth phase involves systems integration and system testing (of programs and procedures)—normally carried out by a Quality Assurance (QA) professional—**to determine if the proposed design meets the initial set of business goals.** Testing may be repeated, specifically to check for errors, bugs and interoperability. This testing will be performed until the end user finds it acceptable.

6. Implementation



The sixth phase is when the majority of the code for the program is written. Additionally, **this phase involves the actual installation of the newly-developed system.** This step puts the project into production by moving the data and components from the old system and placing them in the new system.

7. Operations and Maintenance

The seventh and final phase **involves maintenance and regular required updates.** This step is when end users can fine-tune the system, if they wish, to boost performance, add new capabilities or meet additional user requirements.

Software Development Process

- When you work to build a product or system, it's important to go through a series of predictable steps—a road map that helps you create a timely, high-quality result. The road map that you follow is called a "**Software Process.**"
- **Software Process Model** is an abstract representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

Water Fall Model

Waterfall model is the very first model that is used in SDLC. It is also known as the linear sequential model.

In this model, the outcome of one phase is the input for the next phase. Development of the next phase starts only when the previous phase is complete.

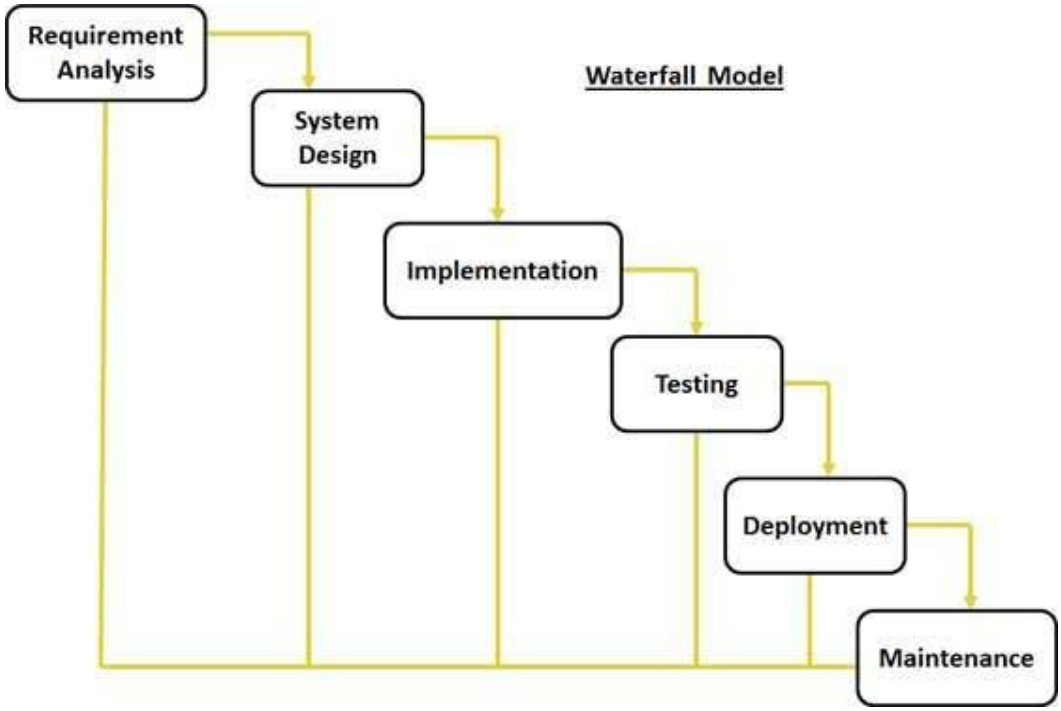


Figure 2.0.7 Waterfall Model

- First, Requirement gathering and analysis is done. Once the requirement is freeze then only the System Design can start.
- In System Design Software Architecture Design, documents which act as an input for the next phase are created.
- In the Implementation phase, coding is done and the software developed is the input for the next phase.
- In the testing phase, the developed code is tested thoroughly to detect the defects in the software. Defects are logged into the defect tracking tool and are retested once fixed.
- In the Deployment phase, the developed code is moved into production after the sign off is given by the customer.

- Any issues in the production environment are resolved by the developers which come under maintenance.

Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

- Simple and easy to understand and use.
- Easy to manage because each phase has specific deliverables.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

- 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-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are changing.

- It is difficult to measure progress within stages.
- Adjusting scope during the life cycle can end a project.

Examples of Waterfall Model

In the olden days, Waterfall model was used to develop enterprise applications like;

- Customer Relationship Management (CRM) systems
- Human Resource Management Systems (HRMS)
- Supply Chain Management Systems
- Inventory Management Systems,
- Point of Sales (POS) systems for Retail chains.
- Development of Department Of Defense (DOD), military and aircraft programs followed Waterfall model in many organizations.
- This is because of the strict standards and requirements that have to be followed.
- In such industries, the requirements are known well in advance and contracts are very specific about the deliverable of the project.

V Model

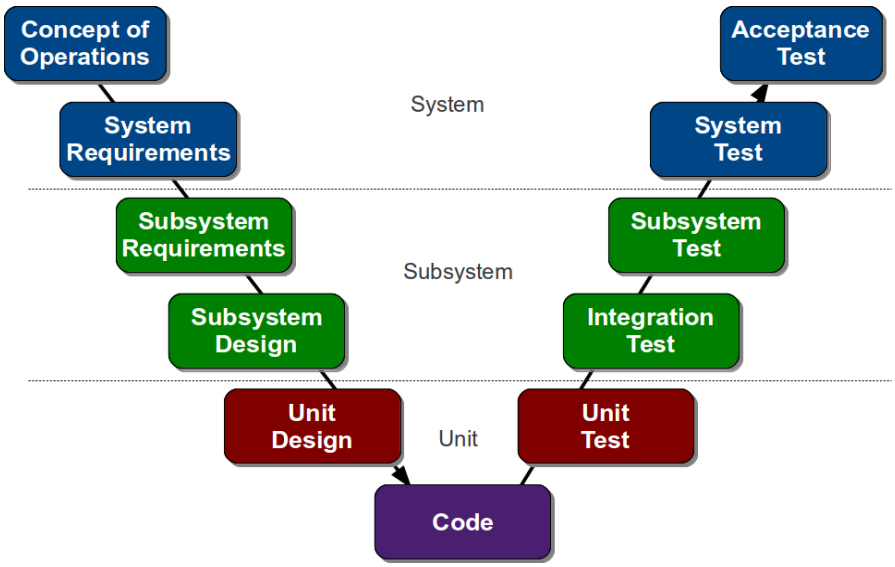


Figure 2.0.8 V Model

V- Model is also known as Verification and Validation Model. In this model Verification & Validation goes hand in hand i.e. development and testing goes parallel.

V model and waterfall model are the same except that the test planning and testing start at an early stage in V-Model.

Following are some of the most suitable scenarios to use the V-Model application

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

V Model - Advantages

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage.

V Model - Disadvantages

- High risk and uncertainty.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are changing.
- Once an application is in the testing stage, it is difficult to go back and change.

Iterative Incremental Process Model

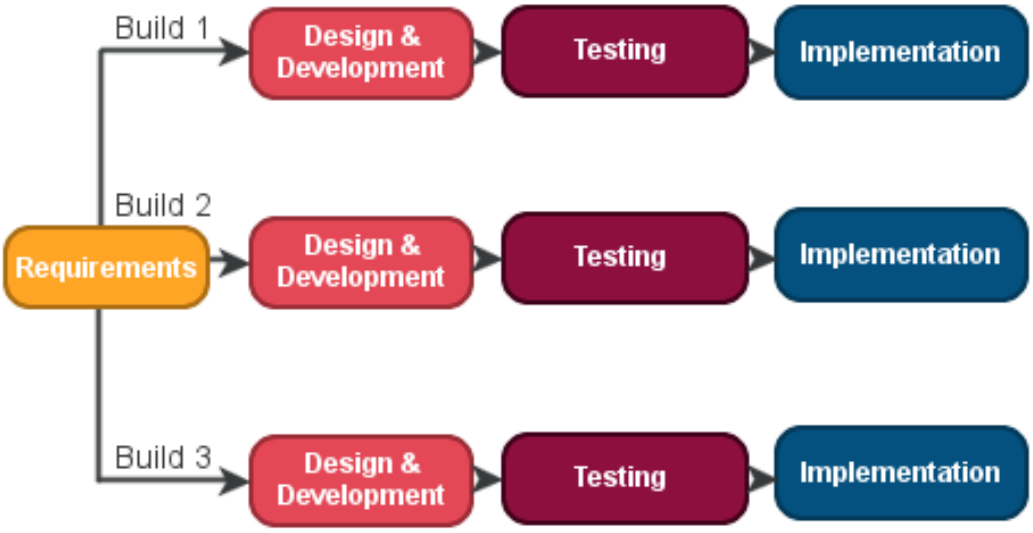


Figure 2.0.9 Iterative Incremental Process Model

The iterative incremental model divides the product into small chunks.

For Example, Feature to be developed in the iteration is decided and implemented. Each iteration goes through the phases namely Requirement Analysis, Designing, Coding, and Testing. **Detailed planning is not required in iterations.**

Once the iteration is completed, a product is verified and is delivered to the customer for their evaluation and feedback. Customer's feedback is implemented in the next iteration along with the newly added feature. Hence, the product increments in terms of features and once the iterations are completed the **final build holds all the features of the product.**

This model is most often used in the following scenarios;

- Requirements of the complete system are clearly defined and understood.
- **Major requirements must be defined;** however, some functionalities or requested enhancements may evolve with time.
- A new technology is being used and is being learnt by the development team while working on the project.
- There are some high-risk features and goals which may change in the future.

Iterative Incremental Process Model – Advantages

- Some working functionality can be developed quickly.
- Results are obtained early and periodically.
- Parallel development can be planned and Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration.
- Easier to manage risk - High risk part is done first.
- With every increment, operational product is delivered.

Iterative Incremental Process Model - Disadvantages

- More resources may be required.
- Although cost of change is lesser, but it is not very 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 life cycle.
- Not suitable for smaller projects.
- Management complexity is more.
- Highly skilled resources are required for risk analysis.

Spiral Model

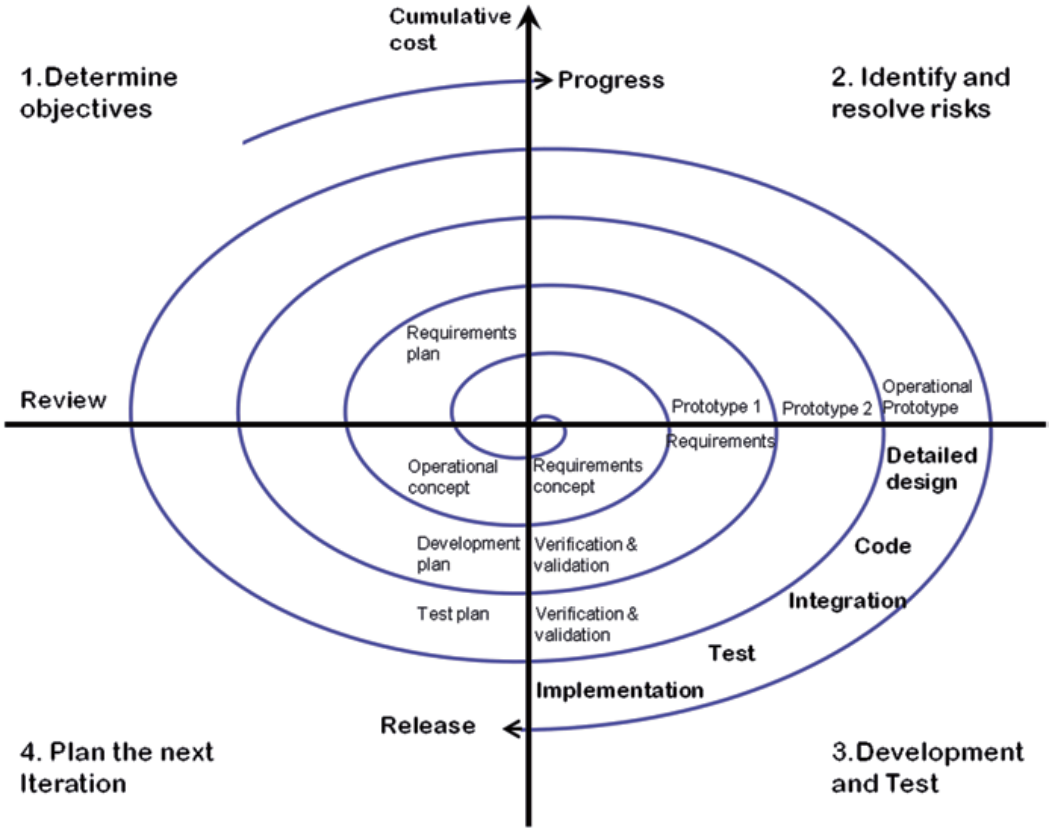


Figure 2.0.10 Spiral Model

The Spiral Model includes iterative and prototype approach.

Spiral Model has four phases:

- **Planning:**

The planning phase includes requirement gathering wherein all the required information is gathered from the customer and is documented.

- **Risk Analysis:**

In this phase, the best solution is selected for the risks involved and analysis is done by building the prototype.

For Example, the risk involved in accessing the data from a remote database can be that the data access rate might be too slow. The risk can be resolved by building a prototype of the data access subsystem.

- **Engineering:**

Once the risk analysis is done, coding and testing are done.

- **Evaluation:**

Customer evaluates the developed system and plans for the next iteration.

Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

This model is most often used in the following scenarios;

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.

Spiral Model - Advantages

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

Spiral Model - Disadvantages

- Management is more complex.
- End of the project may not be known early.

- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex.
- Large number of intermediate stages requires excessive documentation.

Prototype Model

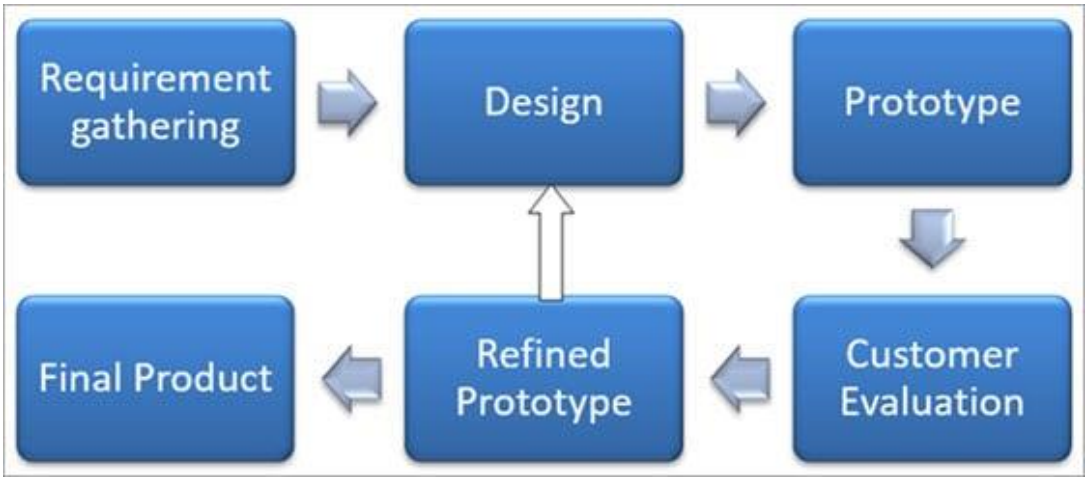


Figure 2.0.11 Prototype Model

The Prototype Model is a model in which the prototype is developed prior to the actual software. Prototype Models have limited functional capabilities and inefficient performance when compared to the actual software. Dummy functions are used to create prototypes. This is a valuable mechanism for understanding the customers' needs. Software prototypes are built prior to the actual software to get valuable feedback from the customer.

Feedbacks are implemented and the prototype is again reviewed by the customer for any change. This process goes on until the model is accepted by the customer. Following Software use to design Prototype quickly,

- AXURE RP:

https://www.axure.com/

- INVISION:

https://www.invisionapp.com/

Prototype Model - Advantages

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

Prototype Model - Disadvantages

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

RAD (Rapid Application Development)

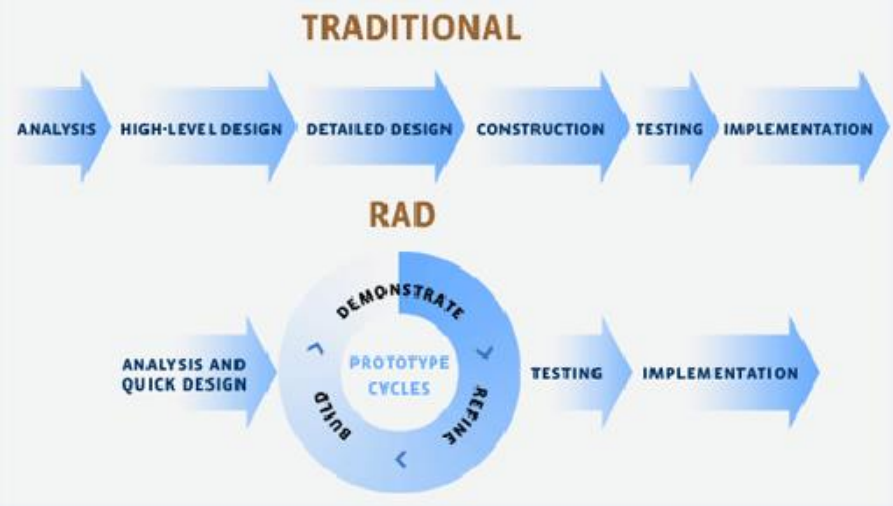


Figure 2.0.12 Rapid Application Development

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved.

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

Lesson 03 – Agile Software Development

What is Agile?

Agile Software Development is a group of Software Development methodologies which based on the Iterative and Incremental development.

It is a conceptual framework that promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. Agile is Rapid, incremental delivery of software by active and continuous communication between developers and customers.

Agile Manifesto

Agile Manifesto is a formal documentation that describe the Agile. The Agile Manifesto is a brief document built on 4 values and 12 principles for agile software development.

Some of the authors formed the Agile Alliance (a non-profit organization) that promotes software development according to the manifesto's principles.

Figure 3.0.1 Agile Manifesto

The agile has 4 overarching values differentiating it from traditional software development processes.

- ***Individuals and interactions*** *over processes and tools*
- ***Working software*** *over comprehensive documentation*
- ***Customer collaboration*** *over contract negotiation*
- ***Responding to change*** *over following a plan*

The Agile Alliance defines 12 agility principles for those who want to achieve agility:

Agile Principle 1

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Agile Principle 2

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Agile Principle 3

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Agile Principle 4

Business people and developers must work together daily throughout the project.

Agile Principle 5

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Agile Principle 6

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile Principle 7

Working software is the primary measure of progress.

Agile Principle 8

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Agile Principle 9

Continuous attention to technical excellence and good design enhances agility.

Agile Principle 10

Simplicity—the art of maximizing the amount of work not done—is essential.

Agile Principle 11

The best architectures, requirements, and designs emerge from self– organizing teams.

Agile Principle 12

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Teams

We are the most important people in AGILE Development.

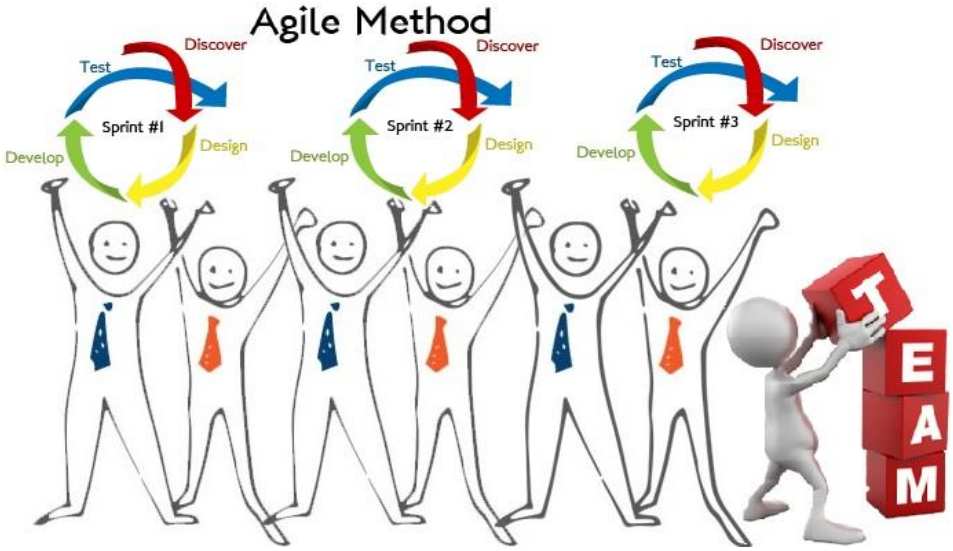


Figure 3.0.2 Most Important people in Agile Development

C<>DE WIZARDS

Figure 3.0.3 Agile Teams

Key Qualities among the People on an Agile Team

Agile team is the most important thing in Agile Development. Normally 6-7 self-organizing, cross functional people are among the team.

- **Competence:** Master in One but knows everything.
- **Common focus:** All are focused on one goal—to deliver a working software increment to the customer within the time promised.
- **Collaboration**: Communication and creating information that will help all stakeholders understand the work of the team.
- **Decision-making ability:** Any good software team (including agile teams) must be allowed the freedom to control its own destiny.
- **Fuzzy problem-solving ability and Self Organize:** Agile team will continually have to deal with issues and must solve those issues in a better manner.

- **Mutual trust and Respect**: The agile team must become "jelled" team. A jelled team exhibits the trust and respect for each other.

Agile Techniques

- Agile Scrum Methodology
- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development (DSDM)
- Crystal
- Feature Driven Development (FDD)
- Test Driven Development (TDD)
- Lean Software Development (LSD)
- Agile Unified Process (AUP)

Scrum Process

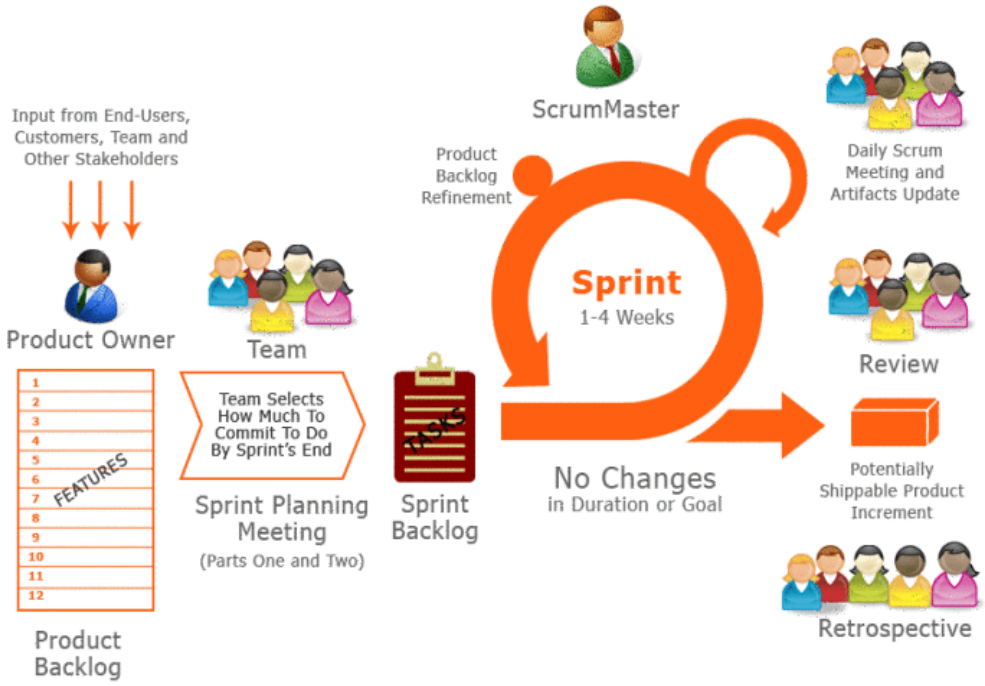


Figure 3.0.4 Scrum Process

Main Roles in Scrum

Scrum Master

Is the person who ensures the process is followed and removes impediment.

Figure 3.0.5 Scrum Master

A good Scrum Master shelters the team from outside distractions, allowing team members to focus maniacally during the sprint on the goal they have selected.

Product Owner or Champion

Product Owner is the Business Analysis. PO represent the stakeholders and the business.

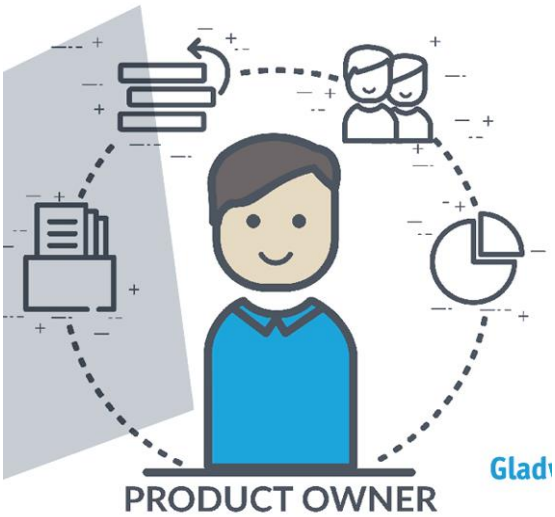


Figure 3.0.6 Product Owner

The Product Owner is responsible for prioritizing the backlog during Scrum development, to ensure it's up to par as more is learned about the system being built, its users, the team and so on.

Development Team

A group who does the coding, implementation, testing etc.

Figure 3.0.7 Development Team

In Scrum, individuals are expected to work beyond their preferred disciplines whenever doing so would be for the good of the team.

Product Back Log & Sprint Back Log

Product Owner get all the requirements and put in the Product Back Log (PB). Those items called as **"User Stories".**

The Product Backlog is a continuously improved list, with the initial version listing only the most preliminary and well-known requirements (no necessary well understood).

Product Backlog evolved based on changes in the product and development environment.

User Stories

Figure 3.0.8 User Story

PRODUCT BACKLOG EXAMPLE

ID	As a...	I want to be able to...	So that...	Priority	Sprint	Status
1	Administrator	see a list of all members and visitors	I can monitor site visits	Must	1	Done
2	Administrator	add new categories	I can allow members to create engaging content	Must	1	Done
3	Administrator	add new security groups	security levels are appropriate	Must	1	Done
4	Administrator	add new keywords	content is easy to group and search for	Must	1	Done
5	Administrator	delete comments	offensive content is removed	Must	1	Done
6	Administrator	block entries	competitors and offenders cannot submit content	Must	1	Done
7	Administrator	change site branding	the site is future-proofed in case brand changes	Could	1	Done
8	Member	change my password	I can keep secure	Must	1	Done
9	Member	update my contact details	I can be contacted by Administrators	Must	2	Work in Progress
10	Member	update my email preferences	I'm not bombarded with junk email	Should	2	Work in Progress
11	Member	share content to social networks	I can promote what I find interesting	Could	2	Work in Progress
12	Visitor	create an account	I can benefit from member discounts	Must		To be started
13	Visitor	login	I can post new entries	Must		To be started
14	Visitor	add comments	I can have a say	Must		To be started
15	Visitor	suggest improvements	I can contribute to the site usability	Should		To be started
16	Visitor	contact the Administrators	I can directly submit a query	Could		To be started
17	Visitor	follow a member's updates	I'm informed of updates from members I find interesting	Should		To be started
18	Visitor	view a member's profile	I can know more about a member	Must		To be started
19	Administrator	generate incoming traffic report	I can understand where traffic is coming from	Must		To be started

Figure 3.0.9 Product backlog example

- During the Sprint Planning Meeting team and Product Owner determine which backlog items go into the sprint.

- During the meeting, the product owner informs the group of the items in the PB that he or she wants to completed and the team then determines how much of they can commit to complete during the sprint and records in the sprint backlog.

Product Backlog Grooming

It is an on-going activity in Sprint rather than a time box event, together with product owners and development teams. Often, development teams have domain knowledge that they can refine themselves.

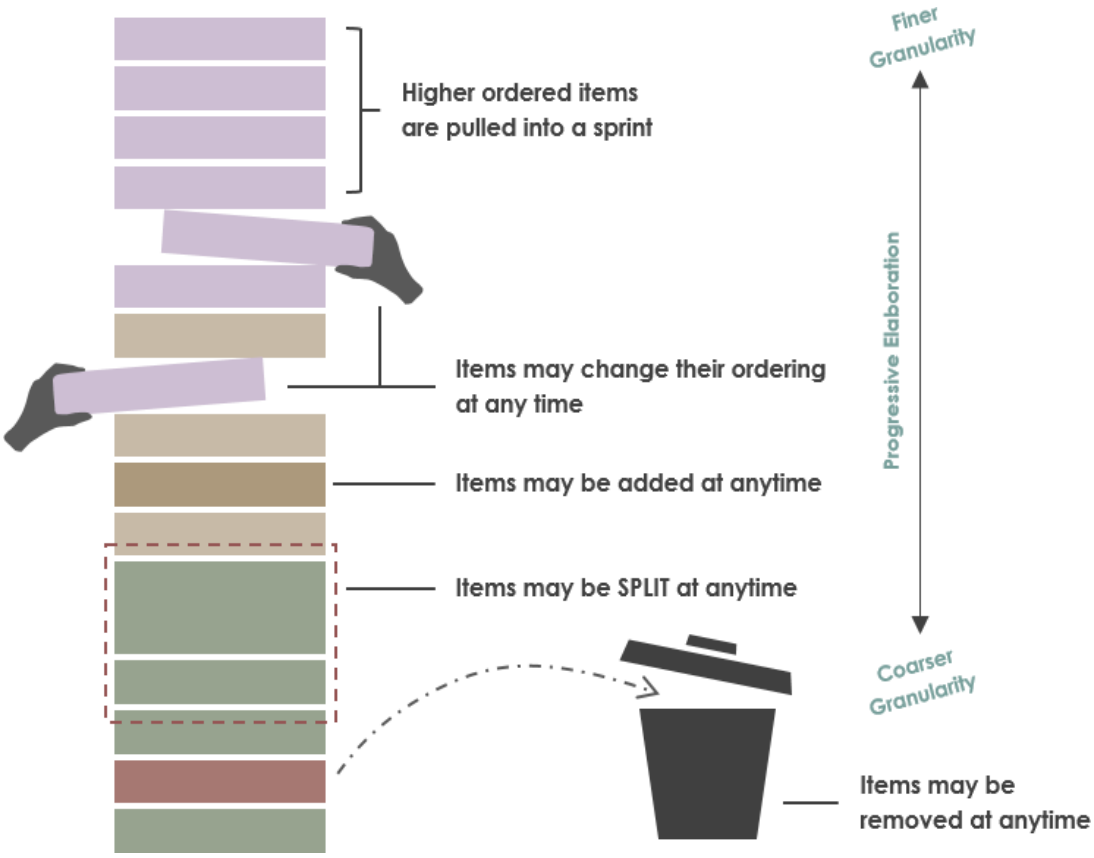


Figure 3.0.10 Product Backlog Grooming

Sprint in the Scrum

Sprint is one time boxed (One week to Four week) iteration of a continuous development cycle. Within a Sprint, planned amount of work has to be completed by the team and made ready for review. Each sprint is preceded by a **Sprint Planning Meeting** where the tasks for the sprint are initiated for the sprint goal.

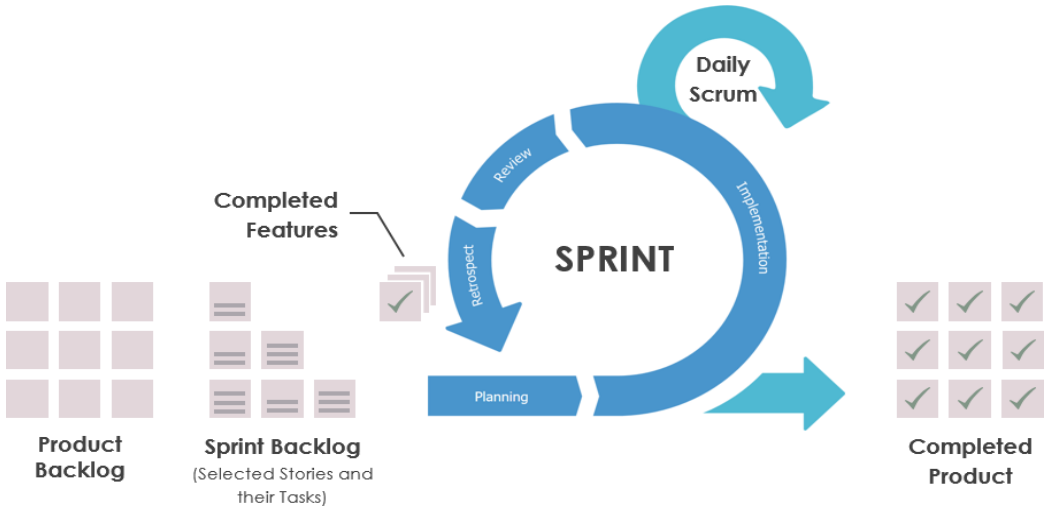


Figure 3.0.11 Sprint

Before the Sprint (Sprint Planning Meeting)

There is a Sprint planning meeting. The length of the sprint planning meeting is proportional to the length of the sprint. This meeting determines what the goals are for that sprint. Based on the team velocity, a set of features are pulled from the top of the product backlog to the Sprint Backlog for the coming Sprint.

Figure 3.0.12 Sprint Planning Meeting

During the Sprint Planning Meeting team and Product Owner determine which backlog items go into the sprint. **"Planning Poker"** is used to estimate how much time is needed for complete the sprint.

Planning Poker

Is used to estimate the effort needed to complete the product backlog items. The product owner need these estimates to find how much time is needed to finish the product backlog items. **"3 Point Estimate"** is used for calculate the most likely time needed for complete the tasks.

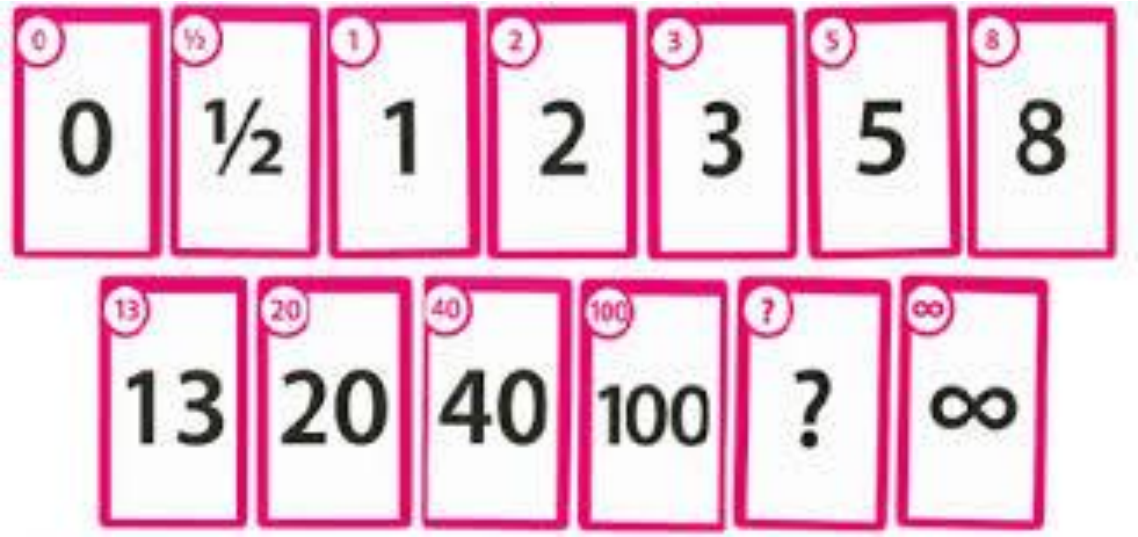


Figure 3.0.13 Planning poker

During Sprint (Daily Scrum Meeting)

No features are added and the sprint goals don't change. The only exception to this is if the team finishes a sprint early. Each day during the sprint, a project status meeting is occurring. This called Daily Scrum or the Daily Standup meeting.

The meeting starts precisely on time. The meeting length is set to 15 minutes. The meeting should happen at the same location and same time every day.

Each person on the team is tasked to answer 3 simple questions:

- What did I do yesterday to help achieve the sprint goal?
- What will I do today to achieve the sprint goal?
- What, if anything is impeding or blocking progress toward the sprint goal?

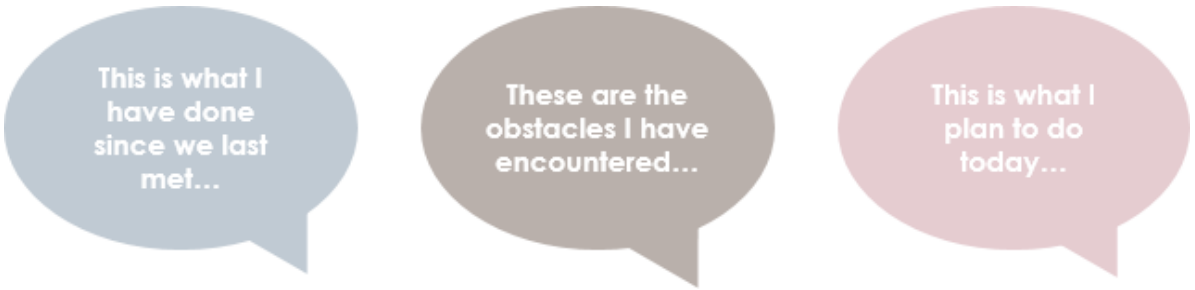


Figure 3.0.14 During Sprint

During Sprint (Task Board / Kanban)

Kanban is Japanese for “visual signal” or “card.” Toyota line-workers used a Kanban to signal steps in their manufacturing process. Scrum task board makes the sprint backlog visible by putting it on a scrum task board.

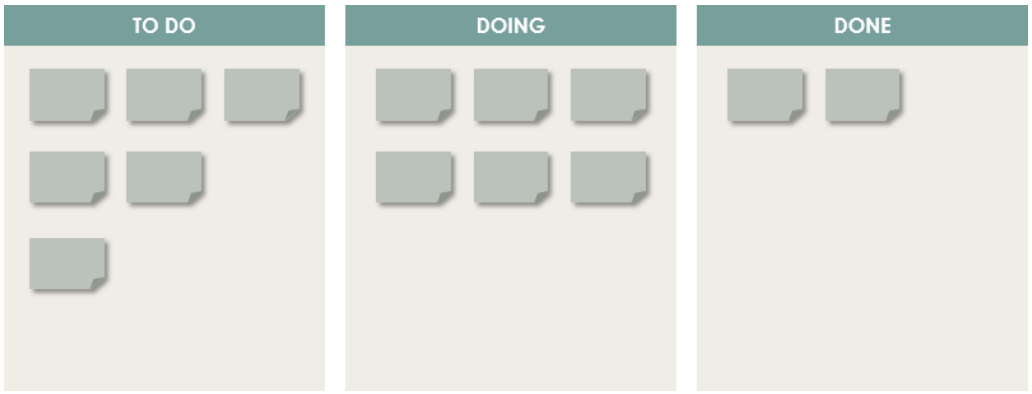


Figure 3.0.15 Task Board / Kanban

Team members update the task board continuously throughout the sprint. Either during or before the daily scrum, estimates are changed and cards are moved around the board. Each row of the scrum task board is a user story.

During Sprint (Sprint Burn Down Chart)

- Daily progress for a sprint over the sprint’s length and This chart showing the remaining work in the sprint backlog.
- Updated every day, it gives a simple view of the sprint progress.
- Must update before go to the daily scrum.

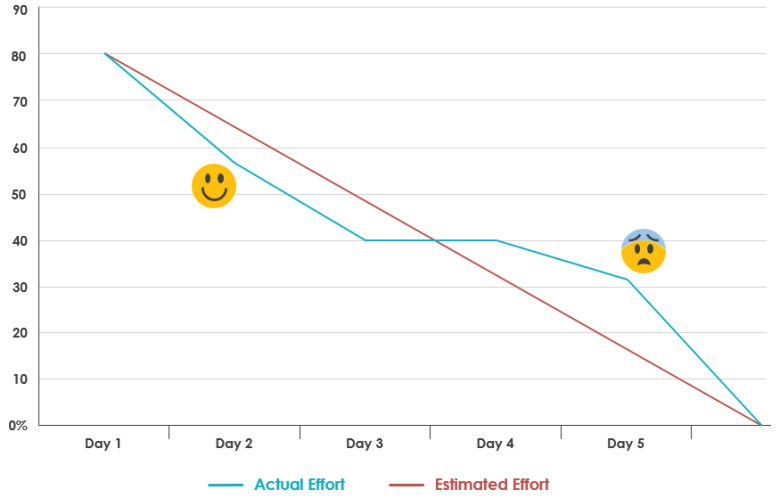


Figure 3.0.16 Sprint Burn Down Chart

Sprint Review Meeting

The sprint review is an informal meeting which the development team, the scrum master, the product owner and the stakeholders will attend. The team gives a demo on the product and will determine what are finished and what aren't.

The purpose of the Sprint Review meeting is for the team to show the customers and stakeholders the work they have accomplished over the sprint and compare it to the commitment given at the beginning of the sprint.

Figure 3.0.17 Sprint Review Meeting

Sprint Retrospective Meeting

The retrospective session is basically an "improvement" meeting held to find ways and means to identify potential pitfalls, past mistakes, and seek out new ways to avoid those mistakes.

The product owner, scrum master, development team members, and optionally stakeholders will be attended.

Two questions asked in the meeting:

- What went well during the sprint?
- What could be improved in the next sprint?

Decide what to start, what to stop and what to continuous.

Figure 3.0.18 Sprint Retrospective Meeting

Lean Software Development

Lean Software Development is an iterative Agile methodology that focuses the team on delivering value to the customer through effective value stream mapping. It is a highly flexible, evolving methodology without rigid guidelines, rules, or methods.

The main principles of the Lean methodology include:

- Eliminating Waste
- Amplifying Learning
- Deciding as Late as Possible
- Delivering as Fast as Possible
- Empowering the Team
- Building Integrity In
- Seeing the Whole

Lean development eliminates waste by asking users to **select only the truly valuable features** for a system, prioritize those features, and then work to deliver them in small batches.

It relies on rapid and reliable feedback between programmers and customers, emphasizing the speed and efficiency of development workflows.

Figure 3.0.19 Main principles of lean methodology

Extreme Programming (XP)

Extreme Programming (XP), is a disciplined approach for high-quality agile software development, focused on speed and continuous delivery. It is intended to improve software quality and responsiveness in the face of changing customer requirements.

It promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals, typically every 1-3 weeks. As an example, code reviews are considered a beneficial practice. Taken to the extreme, code can be reviewed continuously through the practice of **pair programming.**

Figure 3.0.20 Pair Programming

Test Driven Development

- Combination of Test First Development and Refactoring.
- The development team writes tests before they even start to write code.
- They then run the test and if it fails they write the code to ensure the test passes.
- On passing the test, the code is then tied up or refactored and a new test is undertaken.

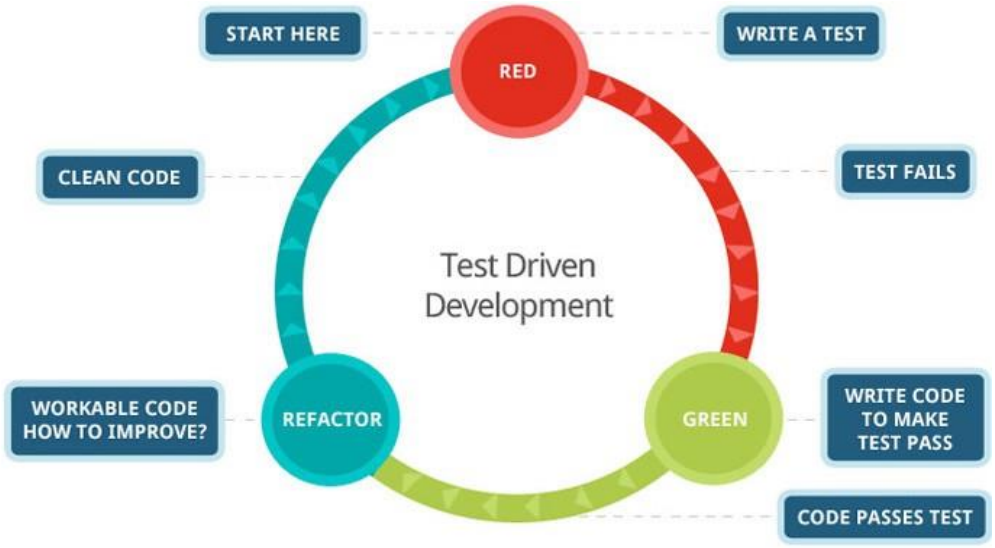


Figure 3.0.21 Test driven development

Dynamic Systems Development Method

This is an Agile Project delivery framework and an Iterative and Incremental. Uses the **MoSCow** prioritization of scope into Musts, Shoulds, Coulds and Won't haves to adjust the project deliverable to meet the stated time constraint.

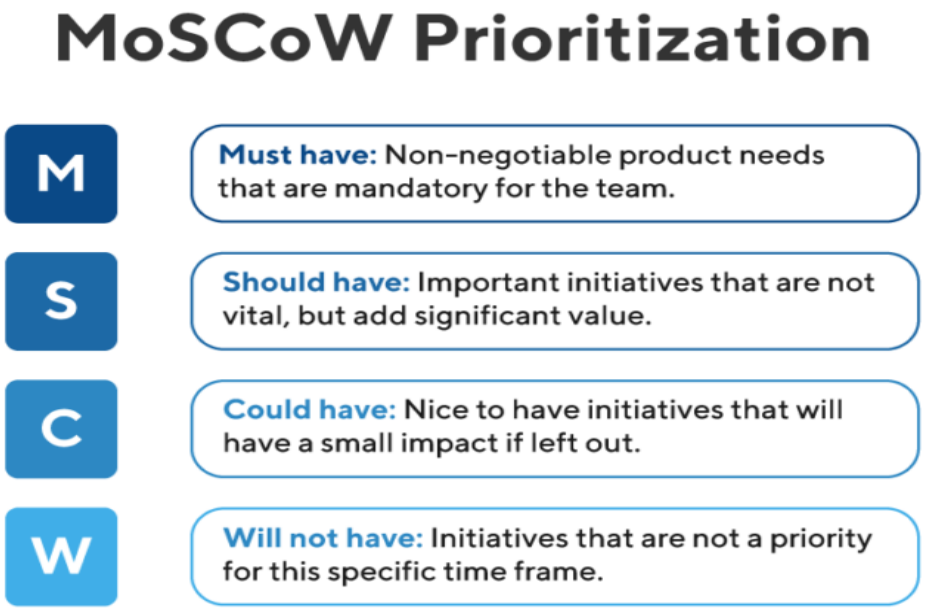


Figure 3.0.22 Dynamic Systems Development Method

Capability Maturity Model Integration (CMMI)

Capability Maturity Model Integration (CMMI) is a framework that describes the key elements of an effective product development and maintenance process. The CMM covers best practice for Planning, Engineering and managing product development and maintenance.

CMMI models provide guidance for developing or improving processes that meet the business goals of an organization. A CMMI model may also be used as a framework for appraising the process maturity of the organization.

Immature vs Mature Software Organization

Immature Software Organizations

- Processes are improvised.
- Managers focus on solving crisis.
- Schedules and budgets are exceeded.
- Product Qualities are bad.
- No objective basis for judging quality.

Mature Software Organizations

- Projects are carried out according to the planned process.
- Mandatory processes are consistent with the way work actually gets done.
- Roles and responsibilities are clear.
- Quality is high.

Characteristics of the Maturity Levels

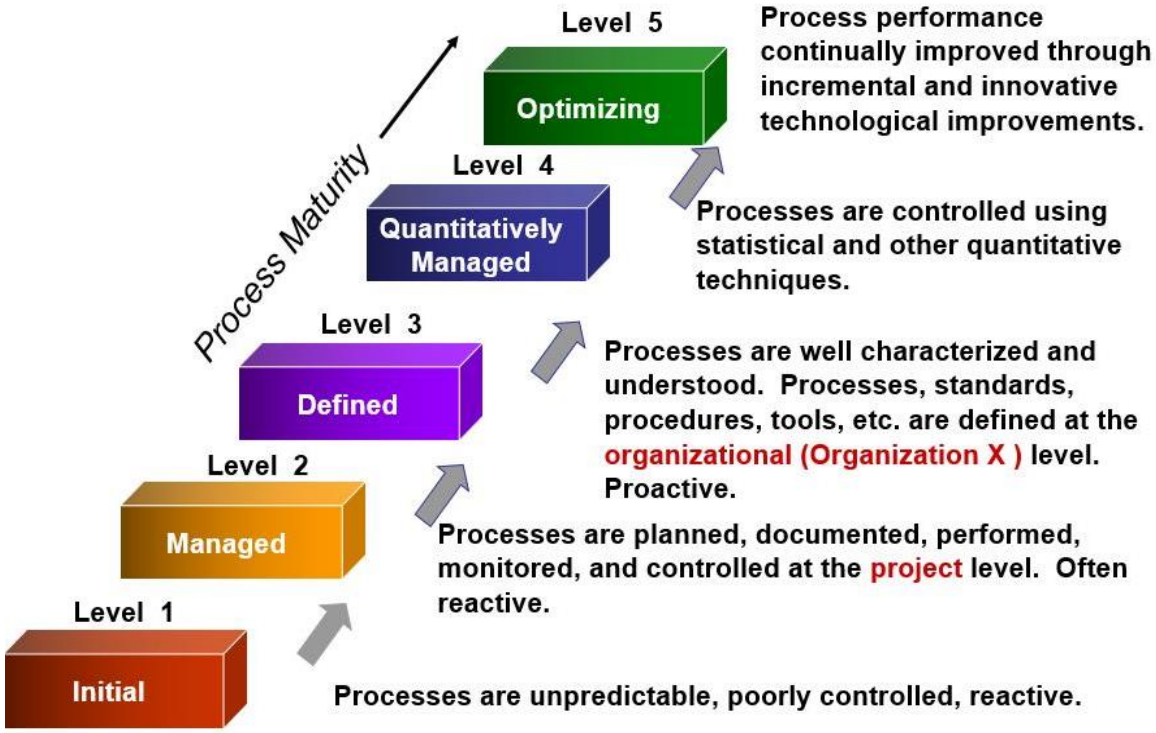


Figure 3.0.23 Characteristics of the Maturity Levels

Lesson 04 – Software Requirement Engineering

Introduction to Software Requirements Engineering

Understanding the requirements of a problem is among the most difficult tasks that face by a Business Analysis and Software Engineer.

The problem is after completing the software,

- Is the system meet the requirements?
- Are the end users have a good understanding of the features and functions?

Requirement engineering establishes a solid base for design and construction. Without it, the resulting software has a probability of not meeting customer's needs.

The following images explain what will happen if you gather the requirements wrongly,

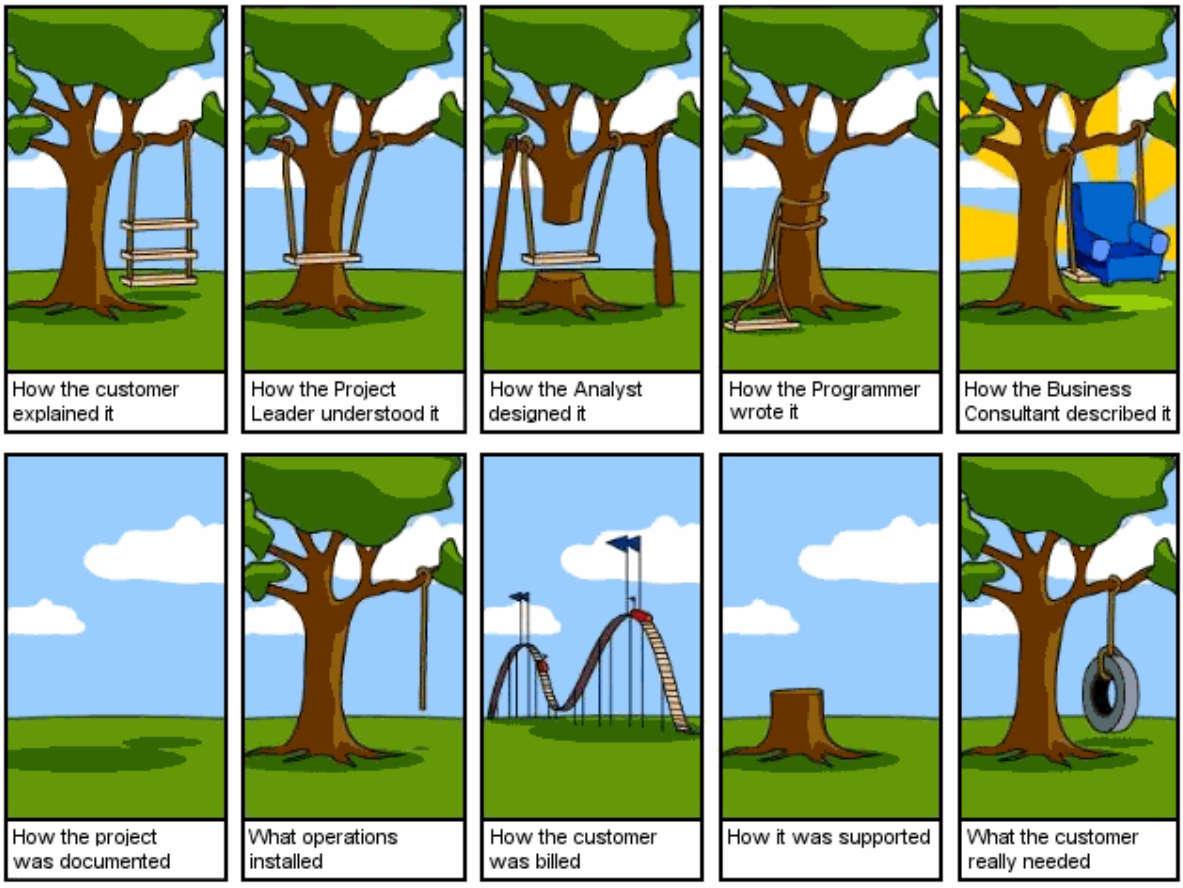


Figure 4.0.1 Wrong Requirement Gathering - 1

Figure 4.0.2 Wrong Requirement Gathering - 2

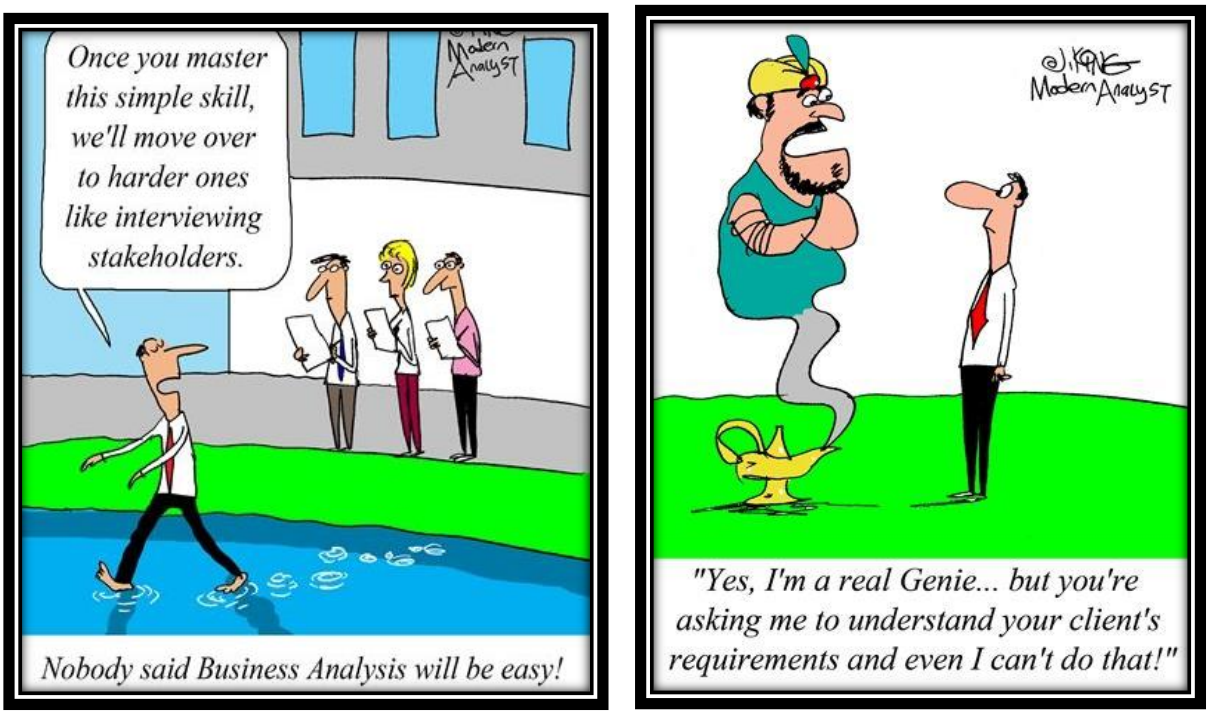


Figure 4.0.3 Wrong Requirement Gathering - 3

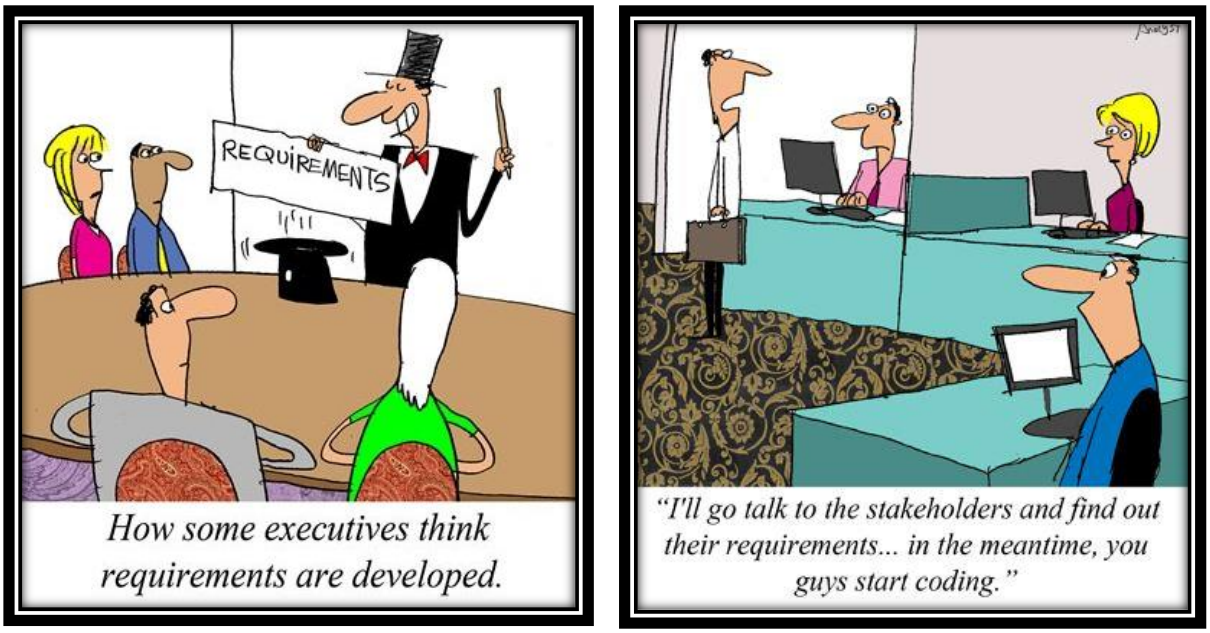


Figure 4.0.4 Wrong Requirement Gathering - 4

Types of Requirements

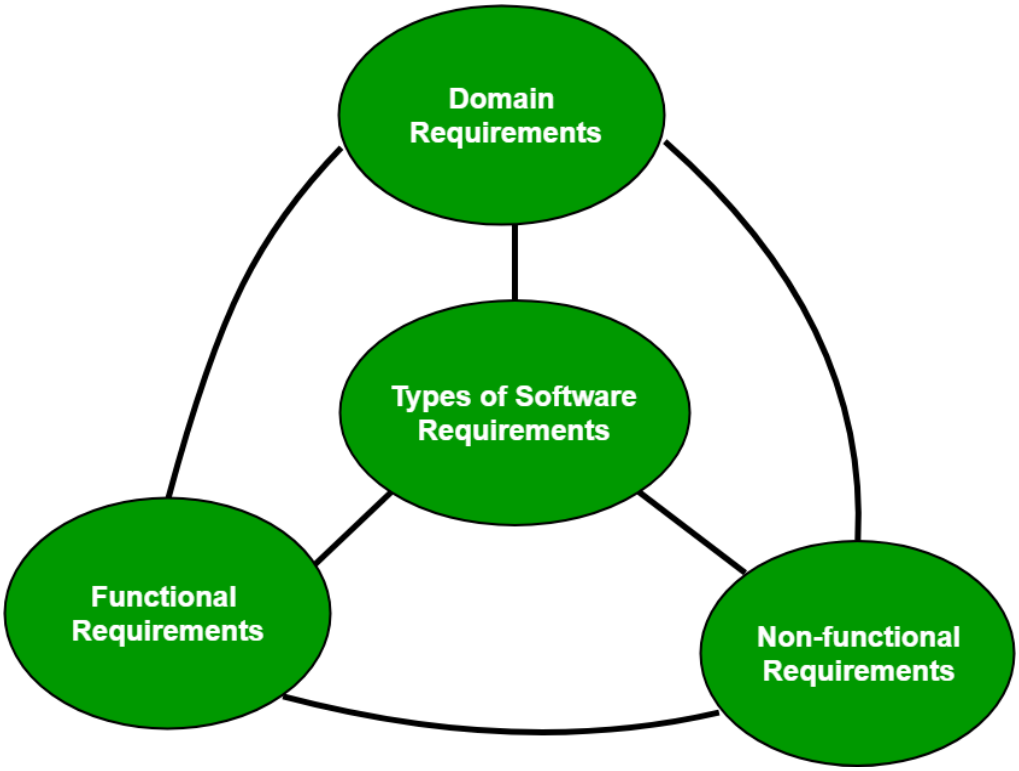


Figure 4.0.5 Types of Requirements

Functional Requirements

- Functionality or services that the system is expected to provide.
- How the System should react to particular inputs and how the system should behave in particular situations.
- Functionality depends on the type of software, expected users and the type of system where the software is used.

  E.g.: The user shall be able to search the relevant data which he needed.

Non-functional Requirements (Quality Requirements)

- Define system properties and constraints such as reliability, response time, interface, security, and storage requirements.
- Non-functional requirements may be categorized in to product requirements, organizational requirements and external requirements.

Non-functional requirements may be more critical than functional requirements.

Types of Non Functional Requirements

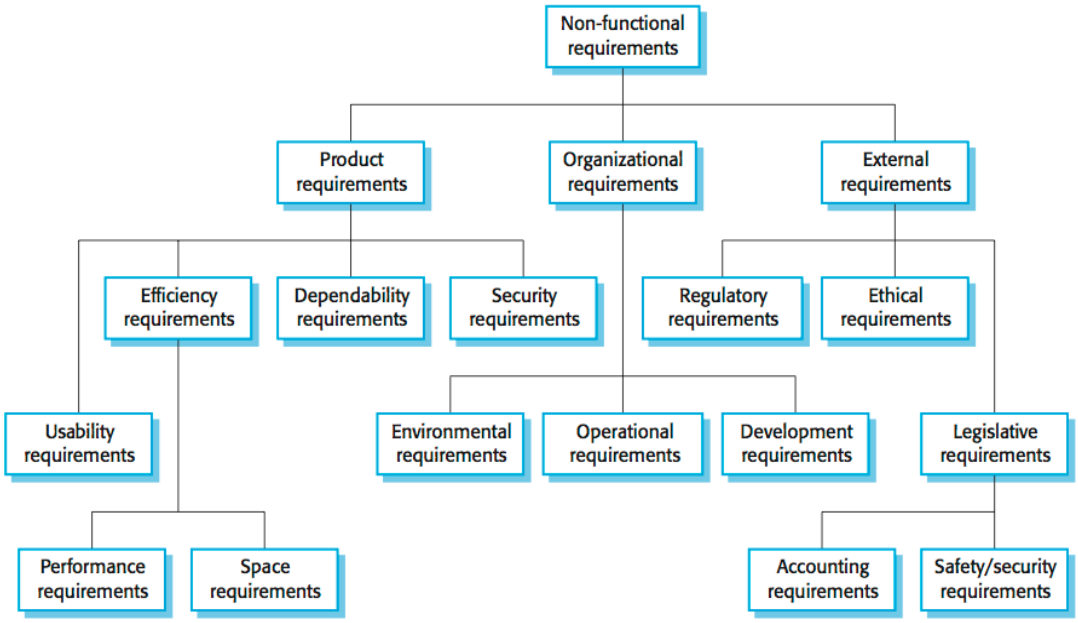


Figure 4.0.6 Types of non-functional Requirements

Product Requirements

Requirements which specify that the delivered product must behave in a particular way.

- Performance Requirements
    - Describe the extent to which the software makes optimal use of resources.
- Reliability Requirements
    - Describe the acceptable failure rate of the software.
- Portability Requirements
    - Describe the ease with which the software can be transferred from one platform to another.
- Interface Requirements
    - Describe the user interface of the software.
- Usability Requirements
    - Describe the ease with which users are able to operate the software.
- Security Requirements
    - Describe the major security areas in the software.

Organizational Requirements

Requirements which are a consequence of organizational policies and procedures like process standards used and implementation requirements.

- Delivery Requirements
    - Specify when the software and its documentation are to be delivered to the user.
- Implementation Requirements
    - Describe requirements such as programming language and design method.
- Standards Requirements
    - Describe the process standards to be used during software development.

External Requirements

Requirements which arise from factors which are external to the system and its development process.

- Interoperability requirements:

Define the way in which different computer based systems will interact with each other in one or more organizations.

- Ethical requirements:

Specify the rules and regulations of the software so that they are acceptable to users.

- Legislative requirements:

Ensure that the software operates within the legal jurisdiction.

Domain Requirements

Requirements come from the application domain of the system and that reflect characteristics of that domain. They may be new functional or non-functional requirements in their own right and constrain existing requirements or set out how particular computation must be carried out.

E.g.: The requirements of the user such as copyright restrictions and security mechanism for the files and documents used in the system are also domain requirements.

Requirement Engineering Process

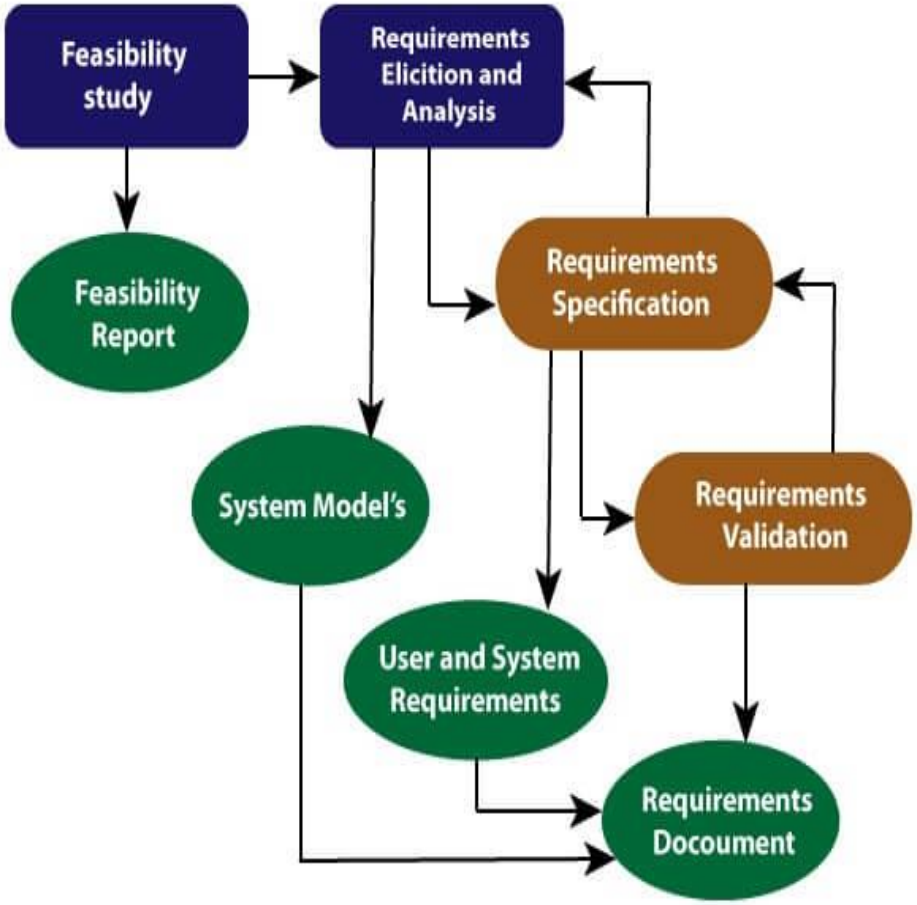
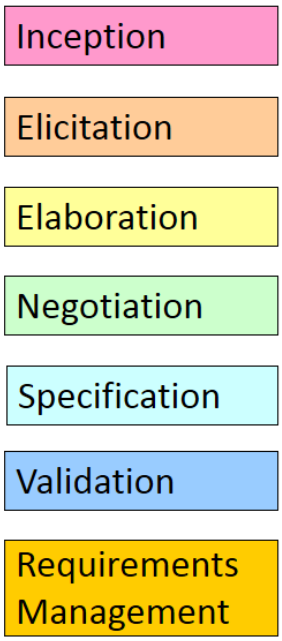


Figure 4.0.7 Requirement Engineering Process

Requirement Engineering Tasks inside the Process

- Some of these tasks may occur in parallel and all are adapted to the needs of the project
- All strive to define what the customer wants
- All serve to establish a solid foundation for the design and construction of the software



1. Inception (Feasibility Study)

- At project inception you establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired.
- When initiating requirements engineering task, the requirements engineer needs to…
  - Identify the stakeholders (Users, Owners, Administrators)
  - Recognize multiple viewpoints
- The output of this phase should be a **Feasibility Study Report** that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

2. Elicitation

- If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user.
- Ask from the Customer, Users and Others
  - What are the objectives for the system or products are?
  - What is to be accomplished?

- How the system or products fits into the needs of the business?
- How the system or product is to be used on a day-to-day basis?

Elicitation Techniques (Requirements Gathering Techniques)

- Brainstorming
- Record Review (Document Analysis)
- Focus Group
- Questionnaires (Survey)
- Observations
- Interview
- Collaborative Requirement Gathering (Joint Application Development)
- Quality Function Deployment
- Prototyping

Brainstorming

Brainstorming can be utilized in requirements gathering to gather a good number of ideas from a group of people. 7Usually brainstorming is used in identifying all possible solutions to problems and simplifies the detail of opportunities.

Record Review (Document Analysis)

Reviewing the current process and documentation can help the analyst understand the business, or system, and its current situation. Existing documentation will provide the analyst the titles and names of stakeholders who are involved with the system.

This will help the analyst formulate questions for interviews or questionnaires to ask of stakeholders, in order to gain additional requirements.

Focus Group

A focus group is a gathering of people who are customers or user representatives for a product to gain its feedback. The feedback can be collected about opportunities, needs, and problems to determine requirements or it can be collected to refine and validate the already elicited requirements.

Questionnaires (Survey)

When gathering information from many people: too many to interview with time constraints and less budget: a questionnaire survey can be used. The survey insists the users to choose from the given options agree / disagree or rate something.

Observations

The observation covers the study of users in its natural habitat. By watching users, a process flow, pain points, awkward steps and opportunities can be determined by an analyst for improvement.

Interview

Interviews of users and stakeholders are important in creating wonderful software. Without knowing the expectations and goal of the stakeholders and users, you are highly unlikely to satiate them. You also have to understand the perspective of every interviewee, in order to properly address and weigh their inputs. **Listening is the most important thing here.**

Collaborative Requirement Gathering (Joint Application Development)

Popularly known as JAD or joint application design, these workshops can be efficient for gathering requirements. The requirements workshops are more organized and structured than a brainstorming session where the involved parties get together to document requirements.

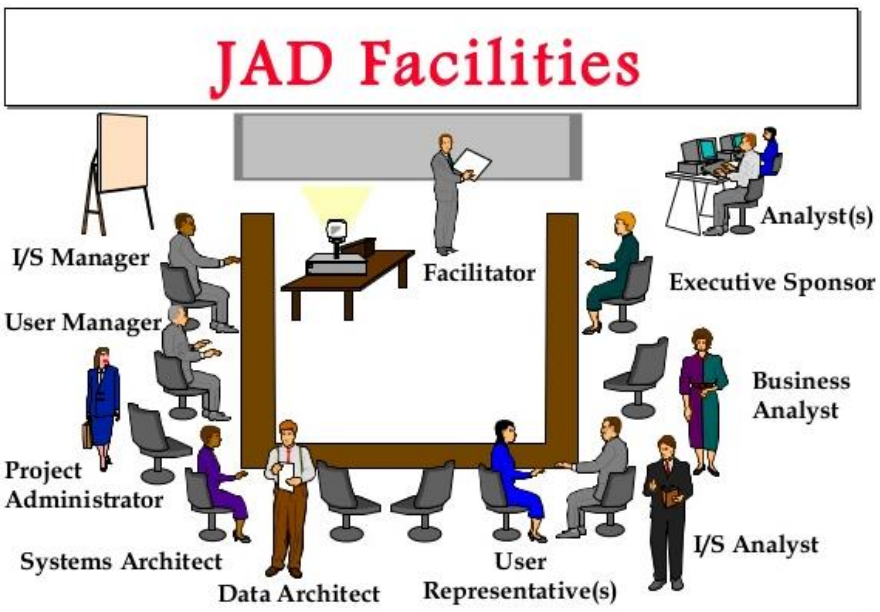


Figure 4.0.8 Joint Application Development

Prototyping

Prototyping can be very helpful at gathering feedback. Low fidelity prototypes make a good listening tool. Many a times, people are not able to articulate a specific need in the abstract. They can swiftly review whether a design approach would satisfy the need. Prototypes are very effectively done with fast sketches of storyboards and interfaces.

3. Elaboration

- During elaboration, the software engineer takes the information obtained during inception and elicitation and begins to expand and refine it.
- Elaboration focuses on developing a refined technical model of software functions, features, and constraints.

4. Negotiation

- During negotiation, the software engineer reconciles the conflicts between what the customer wants and what can be achieved given limited business resources.
- Requirements are ranked (i.e., prioritized) by the customers, users, and other stakeholders.

5. Specification

- The Software Requirement Specification (SRS) is the final work product produced by the Business Analysis or requirements engineer.
- It describes the function and performance of a computer-based system and the constraints that will govern its development.
- It formalizes the informational, functional, and behavioral requirements of the proposed software in both a graphical and textual format.
- It is not a design document. As far as possible**, it should set of WHAT the system should do** rather than HOW it should do.

Software Requirements Specification

Figure 4.0.9 Software Requirements Specification

Software Requirement Specification (SRS) A Template (Karl Wigers)

1. Introduction

    1.1 Purpose of the requirements document

    1.2 Scope of the product

    1.3 Definitions, acronyms and abbreviations

    1.4 References

    1.5 Overview of the remainder of the document

2. General Description

    2.1 Product Perspective

    2.2 Product functions

Writing an SRS in Microsoft Word vs. Helix ALM

You can write your software requirement specification in Microsoft Word. But if a requirement changes, your SRS can fall easily out-of-date. Plus, there can be versioning issues with requirements documents in Word. You can save time — and ensure accuracy — by writing an SRS in **Helix ALM.**

Helix ALM, formerly called **TestTrack**, is application lifecycle management (ALM) software developed by Perforce. The software allows developers to manage requirements, defects, issues and testing during software development.

6. Validation

- During validation, the work products produced as a result of requirements engineering are assessed for quality.
- The specification is examined to ensure that,
  - All the software requirements have been stated unambiguously.
  - All the inconsistencies, omissions, and errors have been detected and corrected.
  - All the requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
  - All the requirements should be practically achievable.

7. Requirements Management

- Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders.
- Each requirement is assigned a unique identifier.
- The requirements are then placed into one or more traceability tables.
- These tables may be stored in a database that relate features, sources, dependencies, subsystems, and interfaces to the requirements.

What is Requirement Traceability Matrix?

- Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases.
- It captures all requirements proposed by the client.
- The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

Figure 4.0.10 Requirement Traceability Matrix

Lesson 05 – Software Design Using Structured Analysis

Structured Analysis

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way. It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes;

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical
- It is an approach that works from high-level overviews to lower-level details.

Structured Analysis Tools:

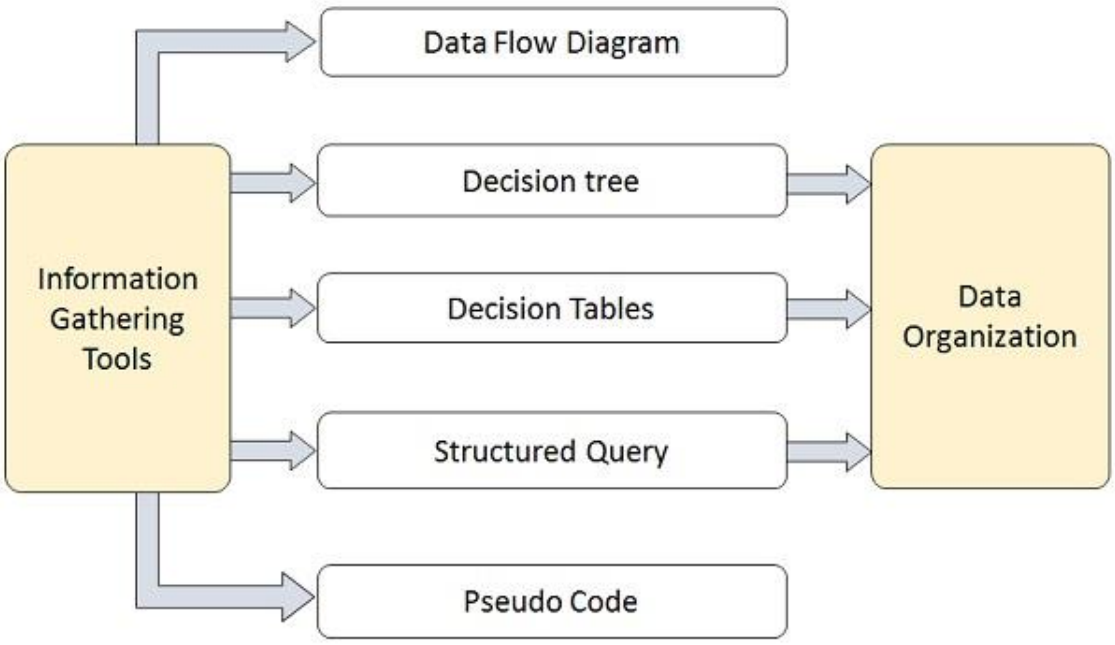


Figure 5.0.1 Structured Analysis

Basically the approach of Structured Analysis is based on the Data Flow Diagram.

Data Flow Diagram (DFD)

- DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system.
- Structure of DFD allows starting from a broad overview and expand it to a hierarchy of detailed diagrams.

Data Flow Diagram Levels

- DFD s are traditionally drawn at three levels.
- The top level DFD is **context diagram** consisting of only one process, representing the entire system.
- Immediately beneath the context diagram is the system level diagram (Level 0). This is a highest level view of the major functions within the system. Further levels can be decomposed to any depth by expanding individual processes
- DFD Levels,
    - Context Diagram
    - Level 0 DFD
    - Level 1 DFD
    - Level 2 DFD

Data Flow Diagram Elements

1. External Entity
2. Processes
3. Data Store
4. Data Flows

1. External Entity

- An external entity may actually be outside of the business (Such as government agents, Customers, Suppliers or Bank)

OR

- It may be inside the business but outside of the system scope (Other Departments, Other internal information systems)
- External entities that supply data into the system are called **Sources**.
- External entities that use system data are called **Sinks or Destination**.

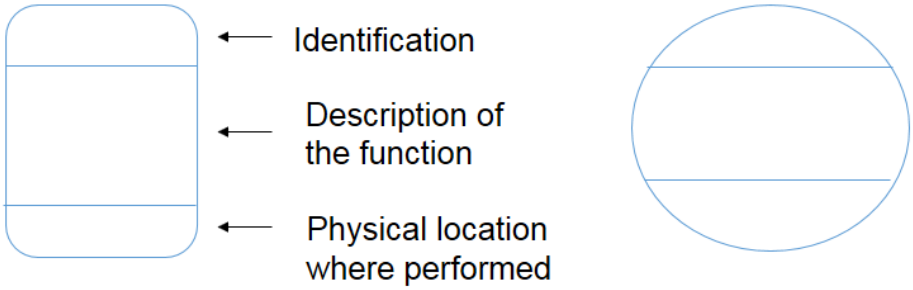
Symbol:

To avoid crossing data flow lines the same entity can be drawn more than once on the same diagram.

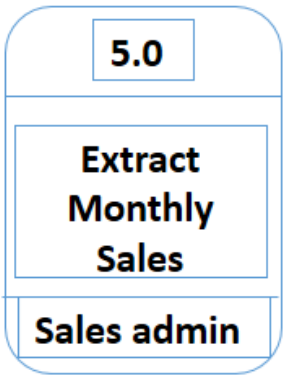
2. Processes

- Any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

Symbol:



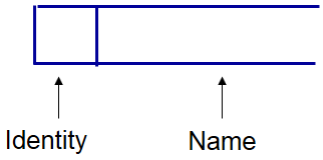
Example:



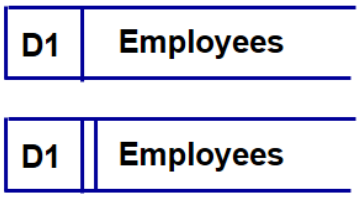
3. Data Store / File

- Files or repositories that hold information for later use, such as a database table or a membership form. Process can enter data in to a data store or retrieve data from a data store. Each data store has a unique name and identity.

Symbol:



Example:

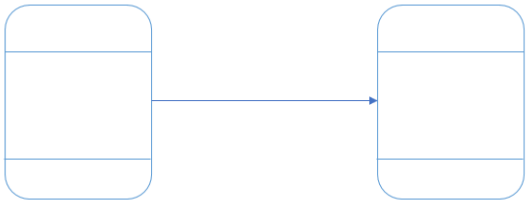


4. Data Flow

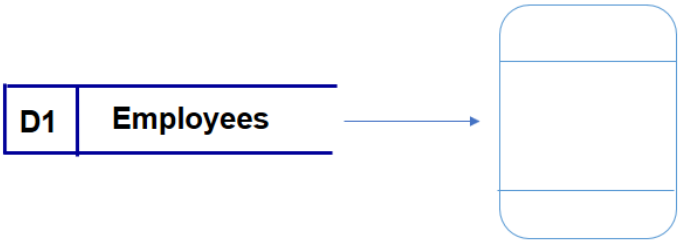
- The route that data takes between the external entities, processes and data stores. The direction of the flow is indicated by an arrow and the line is labeled by the name of the data flow.
- Flow of data in system can take place,
    - Between two processes
    - From a data store to a process
    - From a process to data store
    - From source to a process
    - From process to a sink

Symbol:

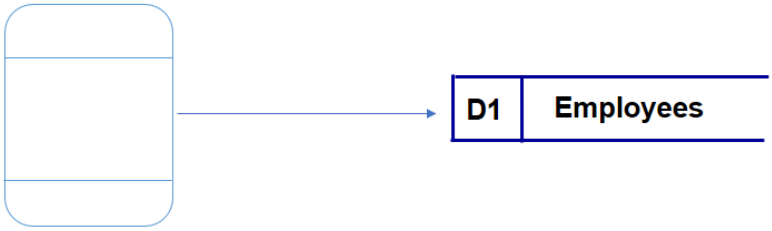
- **Between two processes;**



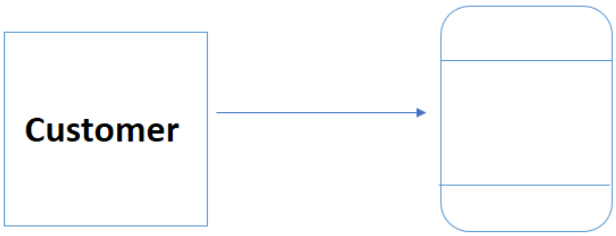
- **From a data store to a process;**



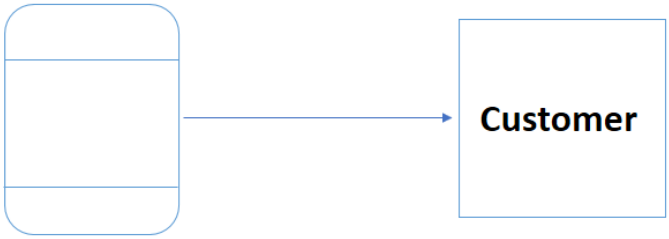
- **From a process to data store;**



- **From source to a process;**



- **From process to a sink;**



Data Flow Diagram RULES

Process

- No process can have only outputs (a miracle)
- No process can have only inputs (black hole)
- A process has a verb phrase label (E.g.: Receive Customer Order)

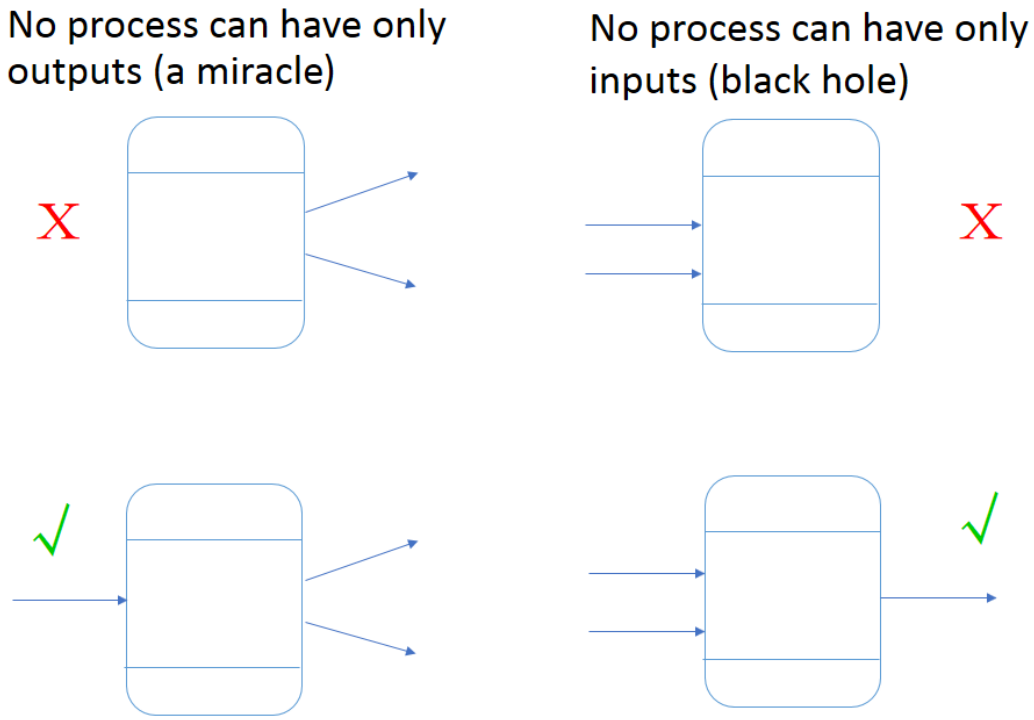


Figure 5.0.2 Data Flow Diagram RULES - Process

External Entity

- Data cannot move directly from one entity to another.
- Data cannot move directly from an entity to data store.
- An entity has a noun phrase label. (E.g.: Customer)
- An entity can be another department, information system, or an organization, that sends or receives data from the system.

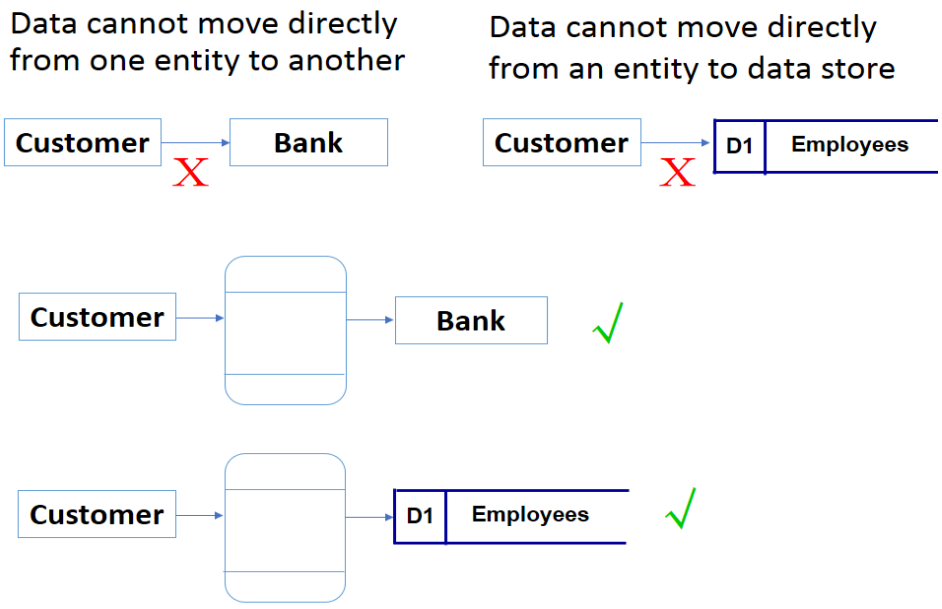


Figure 5.0.3 Data Flow Diagram RULES – External entity

Data Store

- Data cannot be moved directly from one store to another.
- Data cannot move directly from an outside source to a data store.
- Data cannot move directly from data store to a data sink.
- In order to keep the diagram uncluttered, you can repeat data stores on a diagram.
- Data store has a noun phrase label. (Eg: Employee)

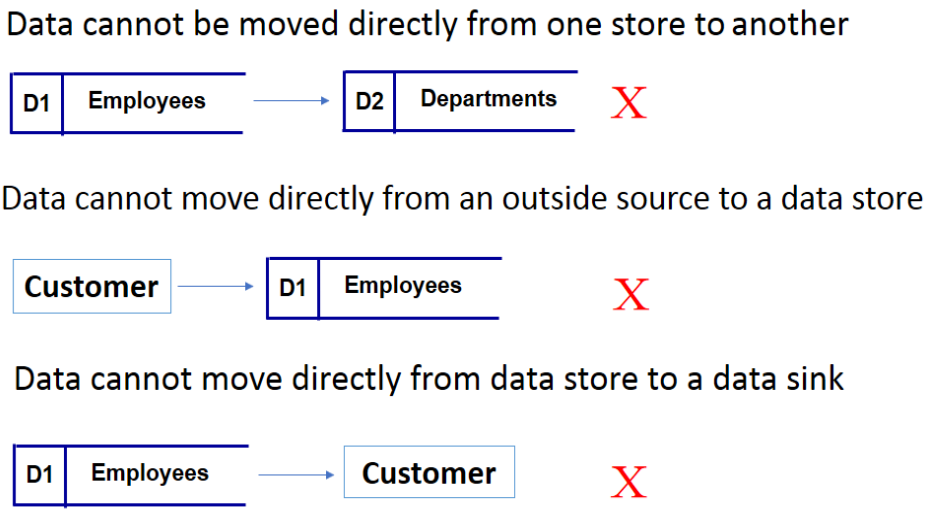


Figure 5.0.4 Data Flow Diagram RULES – Data store – False

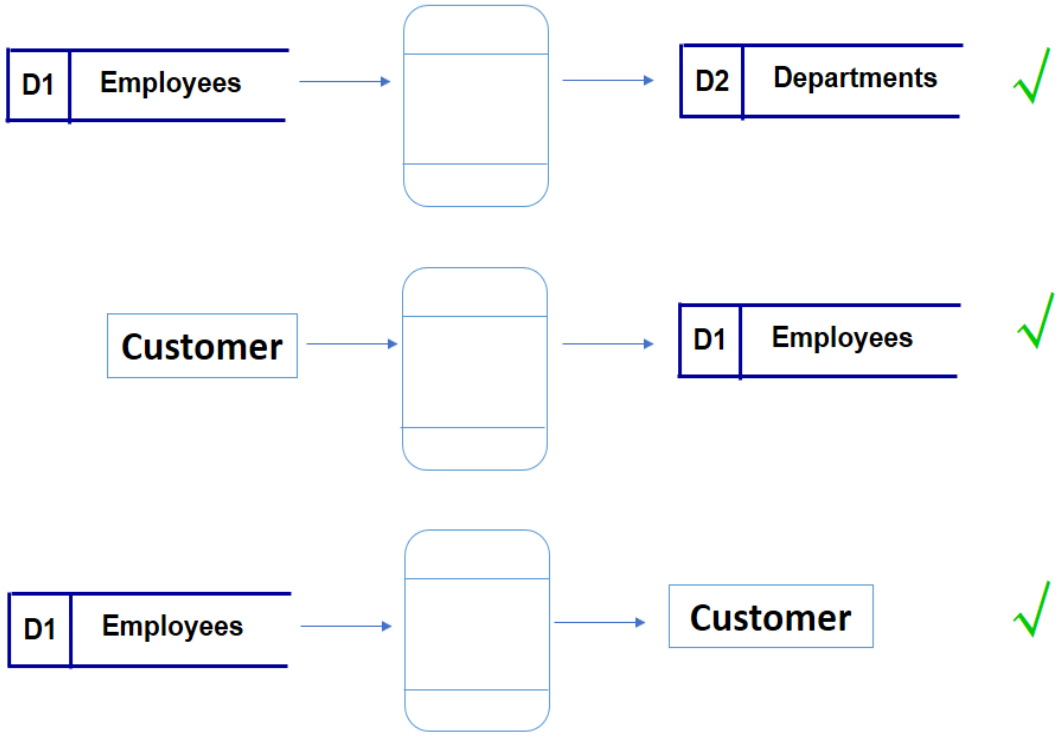


Figure 5.0.5 Data Flow Diagram RULES – Data store – True

- Data flow has only one direction of flow between symbols.
- A fork means that exactly the same data goes from a common location to two or more processes, data stores or sources/sinks.
- A join means that same data comes from any two or more different processes, data stores or entities to a common location.
- A data flow cannot go directly back to the same process it leaves.
- A data flow to a data store means update.
- A data flow from a data store means retrieve or use.
- A data flow has a noun phrase label (Eg: Customer Order)

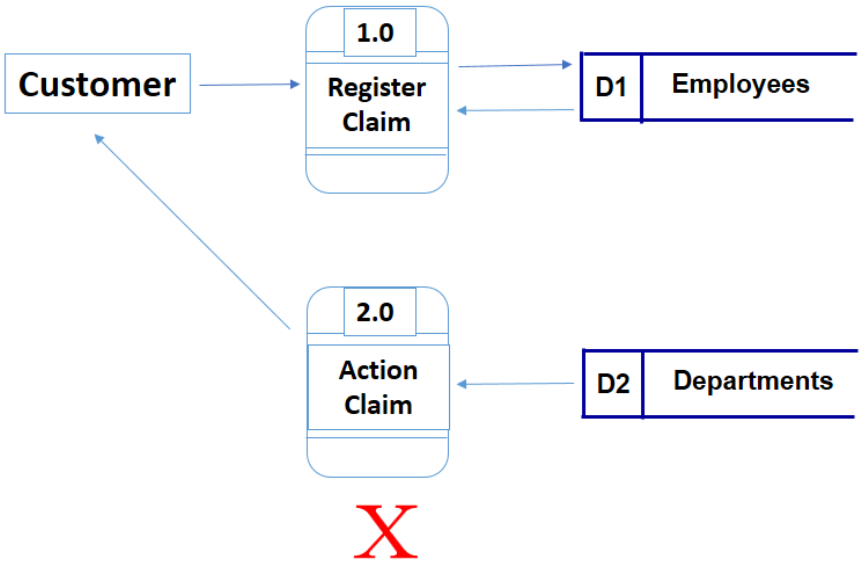


Figure 5.0.6 Data Flow Diagram RULES – Data store – Example

Process 2 must link to Process 1 some way. Either directly with data flow coming from Process 1 to Process 2 or through one data flow going into a data store from Process 1 and another data flow coming out of the data store into Process 2.

Context Diagram

A data flow diagram (DFD) of the scope of an organizational system that shows the system boundaries, external entities that interact with the system and the major information flows between the entities and the system.

- DFD with a highest level of abstraction.
- Represents the whole system as one process (black box).
- Has External Entities.
- Has Data Flows.
- No Data Stores.
- The entire software system is shown as a single process.
- Shows how system interacts with its external entities.

Level 0 Diagram

- Expand the context diagram to show the breakdown of the system.
- Level 0 DFD also mentions basic processes and sources of information.
- It provides a more detailed view of the Context Level Diagram.
- Here, the main functions carried out by the system are highlighted as we break into its sub-processes.

Level 1 Diagram

- Expand the Level 0 diagram to show the breakdown of major processes.
- Level 1 DFD there are a number of data stores, and data-flows between processes and the data stores.
- In Level 1 each process in Level 0 expand to more detail.

DFD Example 1 - Food Ordering System

Hoosier Burger is a restaurant that uses an information system that takes

- Customer orders,
- Send them to the kitchen,
- Bill the customers,
- Monitors goods sold,
- Inventory,
- Generates daily reports (Stock report and Sales Report) to the management.

Context Diagram for Hoosier Burger Food Ordering System

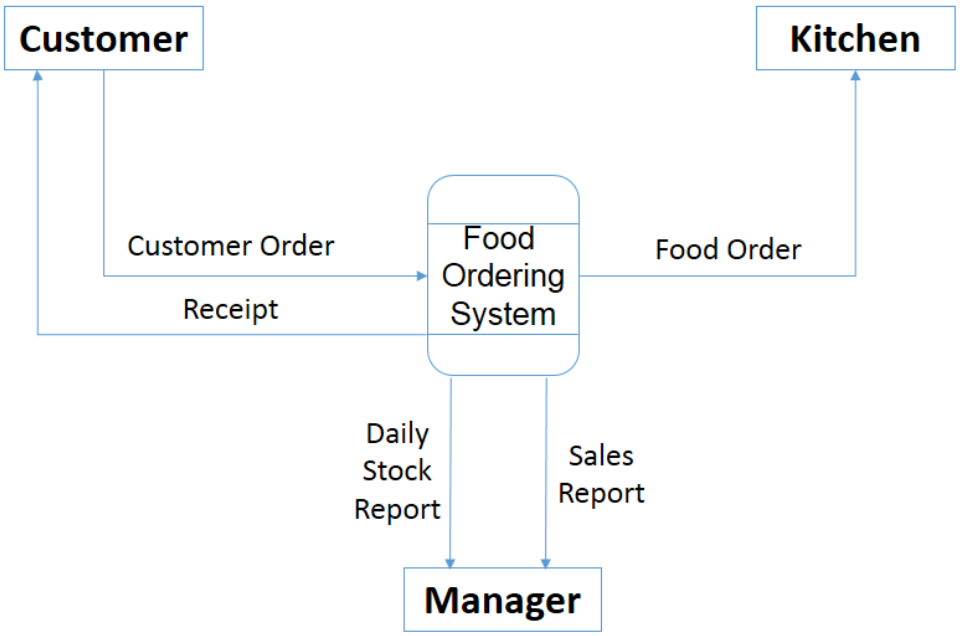


Figure 5.0.7 Context Diagram - Hoosier Burger Food Ordering System

Level 0 for Hoosier Burger Food Ordering System

Expand the context diagram to show the breakdown of major processes.

- **Identify the main possesses of the system**
  - Receive Customer orders
  - Update Inventory file
  - Update Goods sold file
  - Generate Reports
- **Identify the data stores**
  - Generate Reports
  - Inventory file
  - Sold Goods File
  - Customer Orders File
- **Identify the data flows between elements**

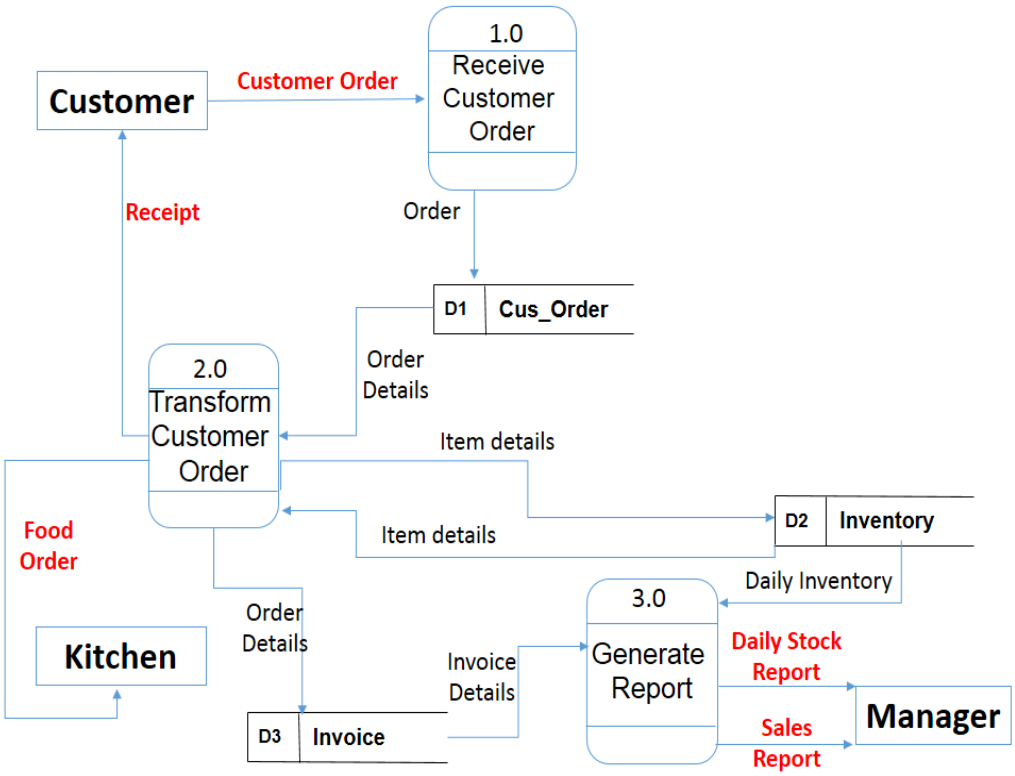


Figure 5.0.8 Level 0 for Hoosier Burger Food Ordering System

In detail way of Level 0 for Hoosier Burger Food Ordering System

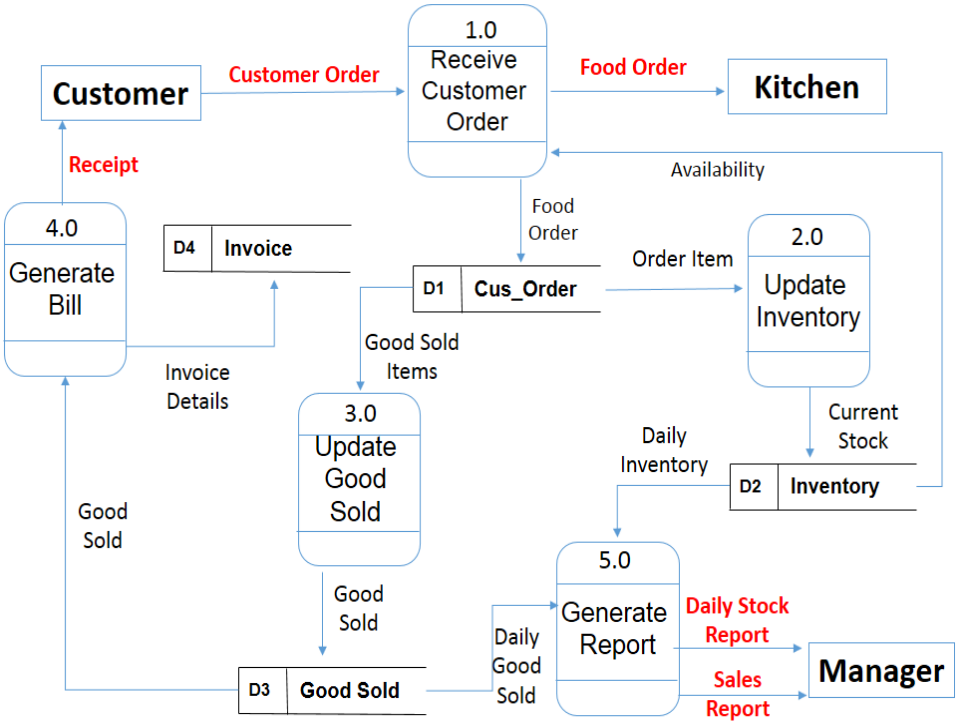


Figure 5.0.9 In detail way of Level 0 for Hoosier Burger Food Ordering System

Level 1 for Hoosier Burger Food Ordering System

Here Generate Reports (5.0) major process we can further sub divide into 2 processes as follow.

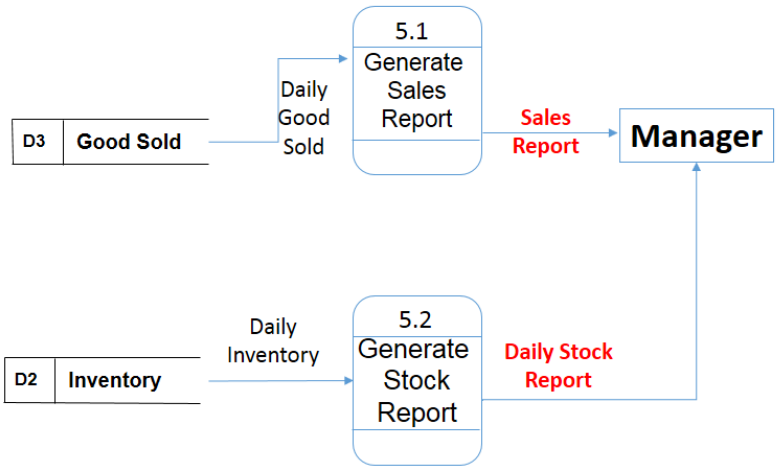


Figure 5.0.10 Level 1 for Hoosier Burger Food Ordering System

Remember you must balance the Context Diagram, Level 0 and Level 1. Also remember you can create processes and data stores depend on the requirements. Above diagrams are not the only corrected diagrams.

As an example in Context Diagram Customer Entity has one input and one output. In Level 0 Customer Entity has only one input and one output. You cannot add new data flow into the Level 0 if the data flow is not available in the Context Diagram.

In the Level 0 Diagram Cus_Order (D1) Data store has only two inputs and two outputs to the Receive Customer Order (1.0) process. Also in the Level 1 we must maintain the number of inputs and number of outputs of Cus_Order (D1) Data store.

DFD Example 2 - Video Ordering System

- A Customer can order a video tape.
- The Video shop will deliver the Video with a delivery note and Invoice.
- System also can send purchase order to the Supplier if the Item is not available.
- Supplier will deliver the goods with a delivery note.

- Also System need to save the Supplier details as well.

Context Diagram for Video Shop System

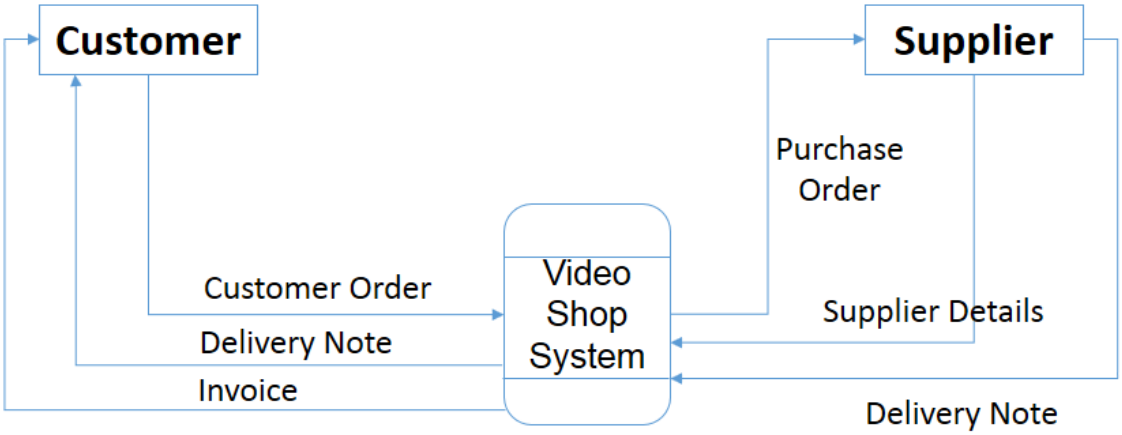


Figure 5.0.11 Context Diagram for Video Shop System

Level 0 for Video Shop System

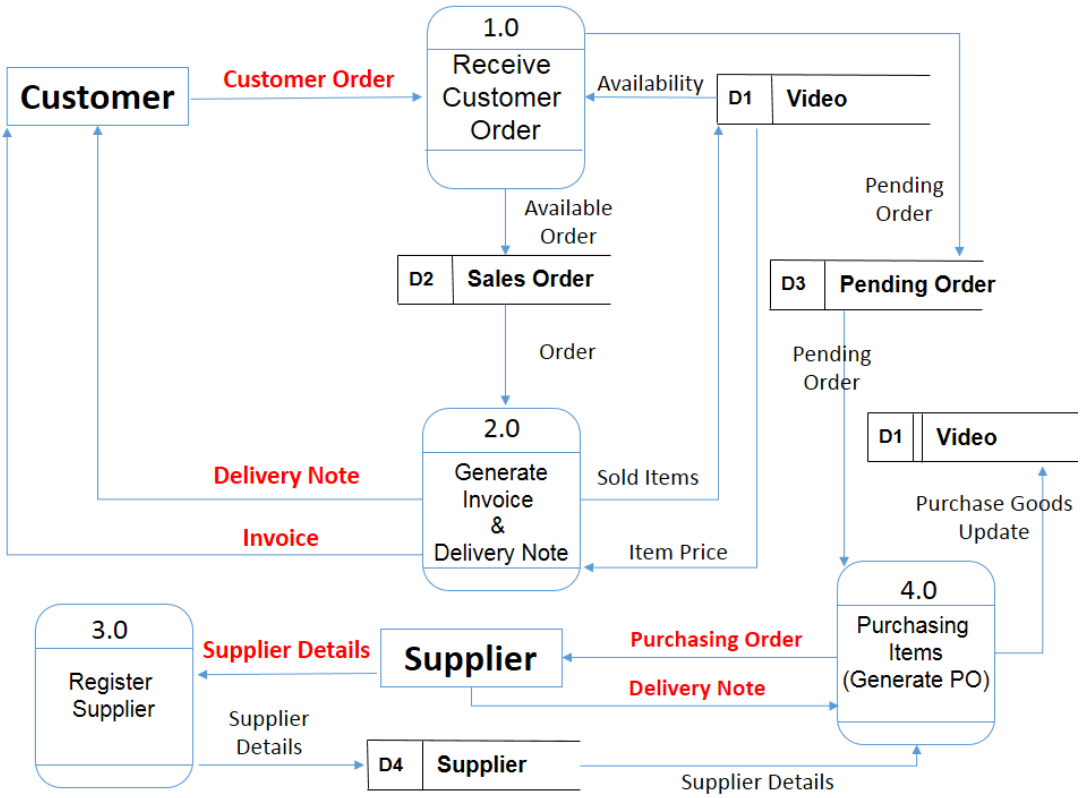


Figure 5.0.12 Level 0 for Video Shop System

Level 1 for Generate Purchase Order Process

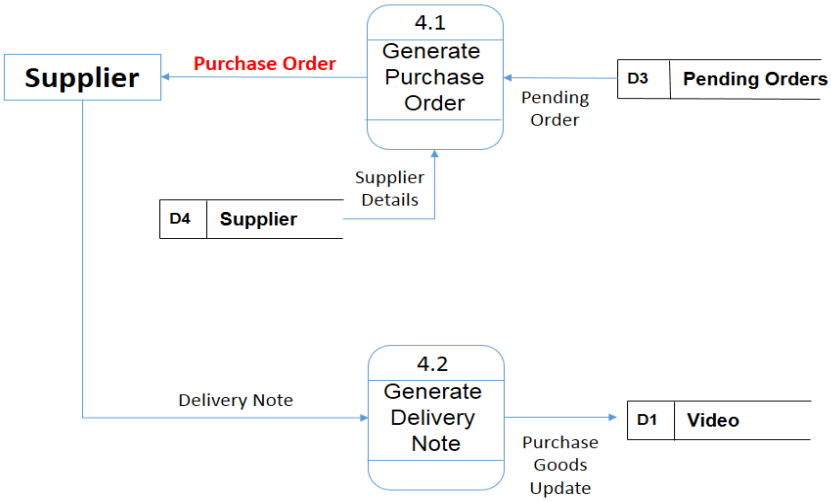


Figure 5.0.13 Level 1 for Generate Purchase Order Process

Please remember balancing the DFD is most important rule in DFD.

You should always check the Data Flow (Input, Output) in Context Diagram and Level 0 Diagram. Then Level 0 and Level 1 Diagram.

DFD Example 3 - Student Management System

- The Staff design courses and provide the course details.
- The Students can enroll to the course through an enrollment request.
- Before starting each course registered students are provided with a course schedule.
- On the starting date of the course, registered students list should be sent to the course director.
- Unavailability of a requested course should also inform to the student.

DFD Example 4 Order Processing System

- Customer can register with the company by sending his/her details to Order Processing system.
- Customer send orders to the Company. And the company sends the product availability. The system accepts orders only if the sufficient inventory exists.
- If the product is available Invoice is generated and send it to the Customer. At the same time Inventory files and Goods Sold files should be updated.
- If the product is not available those orders should save in pending orders File. Then those orders with the item details should send to supplier through a purchase order. Before send the orders to the Supplier, Suppliers should register in the system.
- Once the supplier delivers the product purchasing department get the notification from the system and update the inventory file. At the same time Supplier sends the Delivery notice and Purchasing Department sends the Goods Receive Note (GRN) to the Supplier.
- Each Purchase order and Delivery Note of the Supplier need to save.
- If the products are damaged customer will return the products. Also if the products are damaged company will return the products back to the supplier. Those damage items can be save in Damage Item File.
- Monthly sales report and monthly stock report needs to be send to the Manager.

Lesson 06 – Software Design Using Object Oriented Analysis (Use Case Diagrams)

Object Oriented Analysis and Design

Object-Oriented Programming concepts originate from a broader approach to the entire software development life cycle known as object-oriented analysis and design (OOAD). **OOAD is a technical method of analyzing and designing an application based on that system's object models (the logical components of the system that interact with one another).**

Object-Oriented analysis is a process that groups items that interact with one another, typically by class, data or behavior, to create a model that accurately represents the intended purpose of the system as a whole.

Unified Modeling Language (UML)

- The Unified Modeling Language (UML) is a graphical language for Object-Oriented analysis **that gives a standard way to write a software system's blueprint.** It helps to visualize, specify, construct, and document the artifacts of an object-oriented system.
- It is used to depict the structures and the relationships in a complex system.
- UML model describes what a system is supposed to do.

Types of UML Diagrams

The two broadest categories that encompass all other types are Behavioral UML diagram and Structural UML diagram.

Behavioral UML Diagram

- Activity Diagram
- **Use Case Diagram**
- Interaction Overview Diagram
- Timing Diagram

- State Machine Diagram
- Communication Diagram
- **Sequence Diagram**

Structural UML Diagram

- **Class Diagram**
- Object Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram
- Profile Diagram

Behavioral UML Diagram

UML behavioral diagrams visualize, specify, construct, and document the dynamic aspects of a system. The most important behavioral diagrams are **Use Case Diagrams and Sequence Diagrams.**

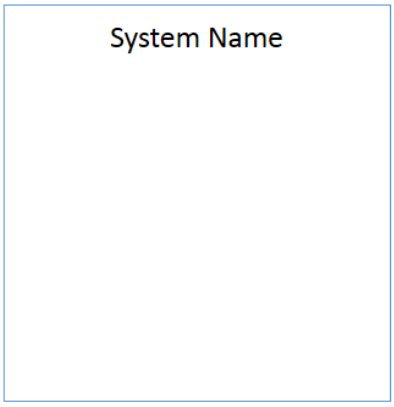
Use Case Diagram

- A use case describes the sequence of **actions (methods)** a system performs yielding visible results. It shows the interaction of things outside the system with the system itself.
- Use case diagram help to understand **how a system should behave.**
- This helps you to gather **requirements from users' point of view.**

Use Case Diagram Notations and Symbols

System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.



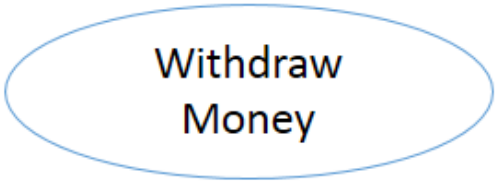
Use Case

System function (automated process or manual process)

Named by verb + Noun (or Noun Phrase).

E.g.: Withdraw Money

Each Actor must be linked to a use case, while some use cases may not be linked to actors.



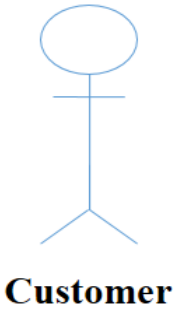
Actor

Someone interacts with use case (system function).

Named by noun. Ex: Customer or Supplier

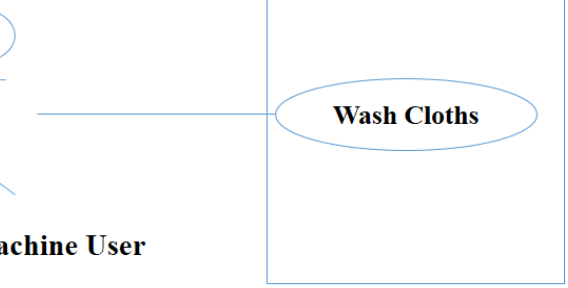
Actor triggers use case(s).

Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).



Relationship (Communication Link)

The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.



Use Case Diagram Example

Figure 6.0.1 Use Case Diagram Example

Different types of Relationship

- Association between Actor and the Use Case
- Include between two Use Cases
- Extends between two Use Cases
- Generalization of a User Case
- Generalization of an Actor

Association between Actor and the Use Case

This one is straightforward and present in every use case diagram.

- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case.

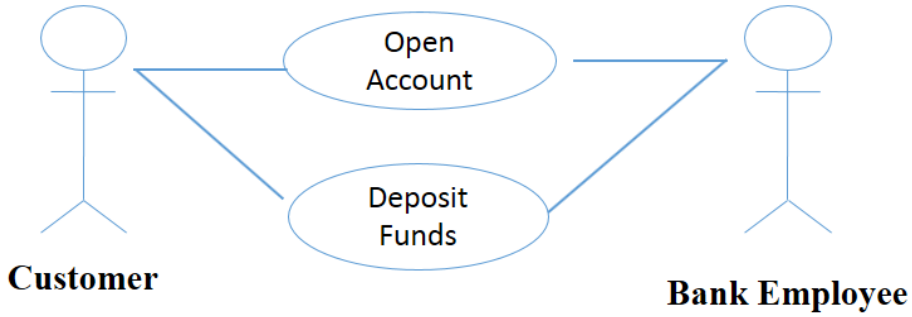


Figure 6.0.2 Association between Actor and the Use Case

Include

When a use case is depicted as using the functionality of another use case. A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behavior as specified in the child use case.

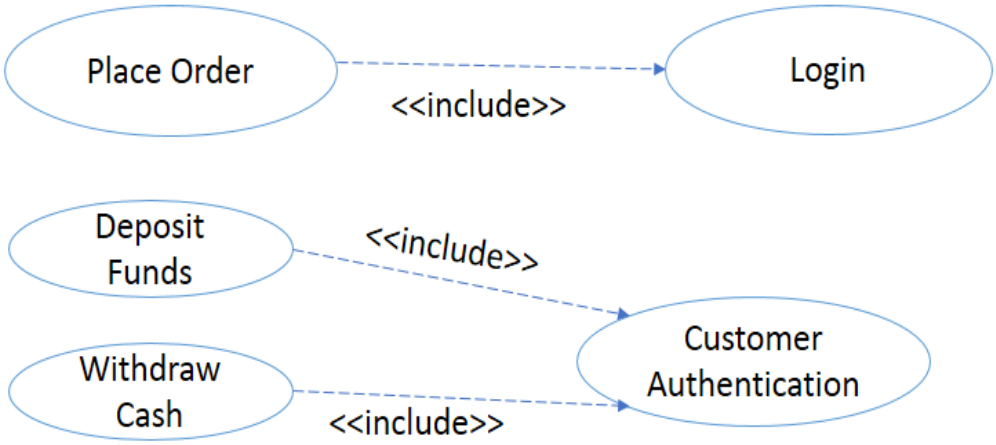


Figure 6.0.3 Include

Extend

Is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional)? The extending use case is dependent on the extended (base) use case.

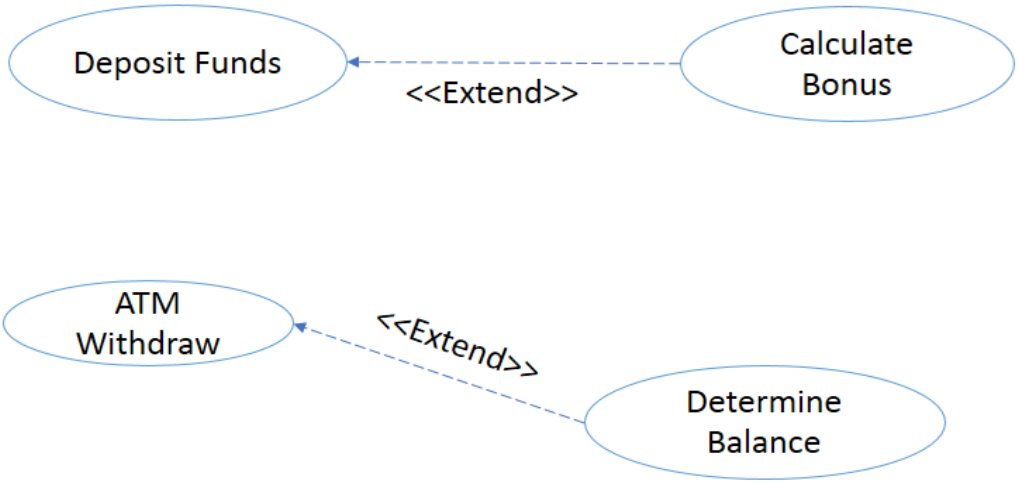


Figure 6.0.4 Extend

Generalization of a Use Case

The relationship which can exist between two use cases and which shows that one Use Case (child) inherits the structure, behavior, and relationships of another actor (parent).

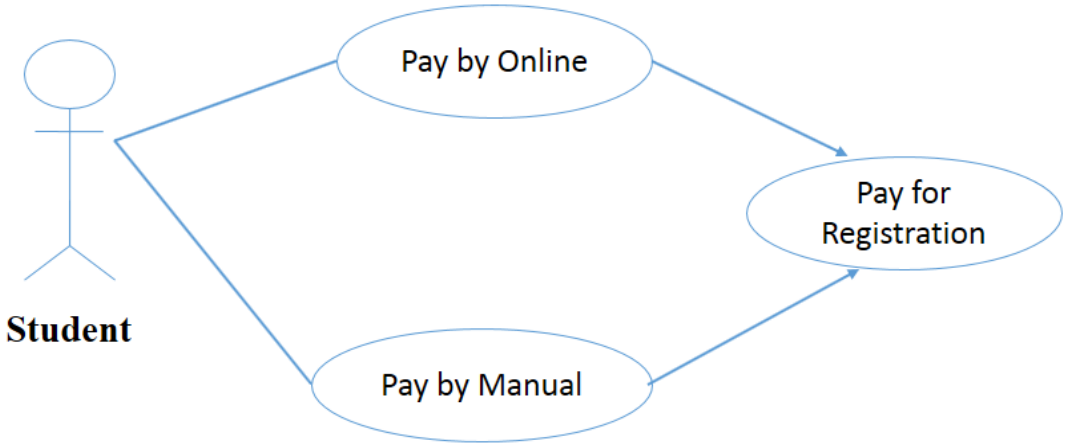


Figure 6.0.5 Generalization of a Use Case

Generalization of an Actor

Generalization of an actor means that one actor can inherit the role of the other actor.

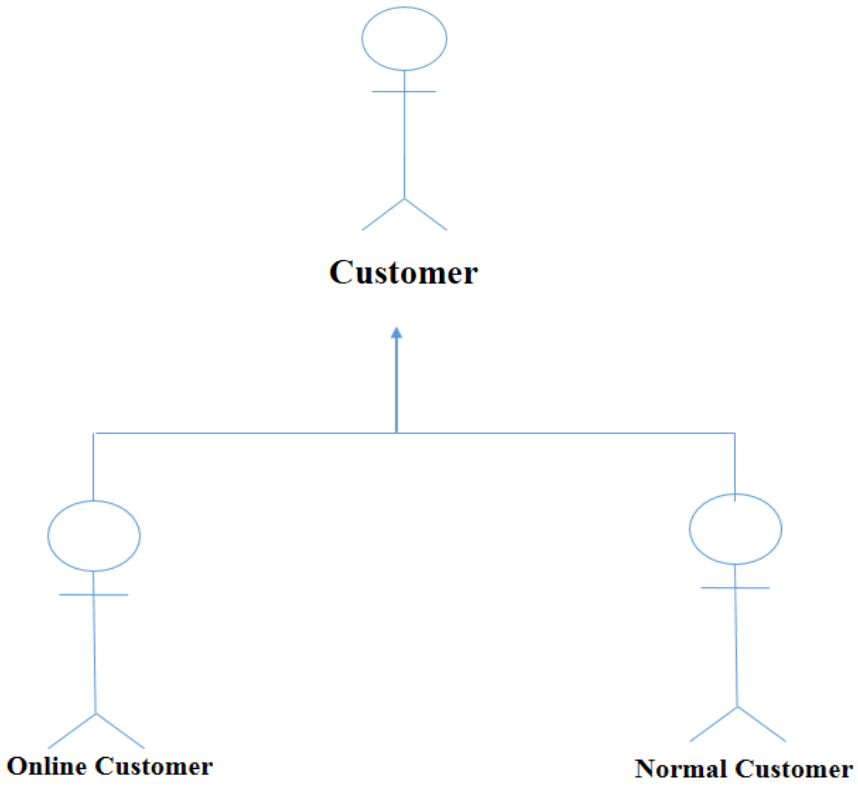


Figure 6.0.6 Generalization of an Actor

Include Relationship

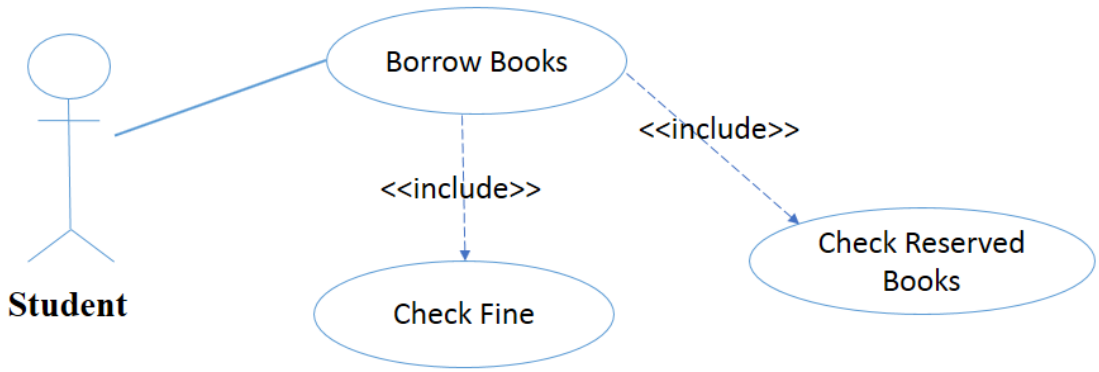


Figure 6.0.7 Include Relationship

Extend Relationship

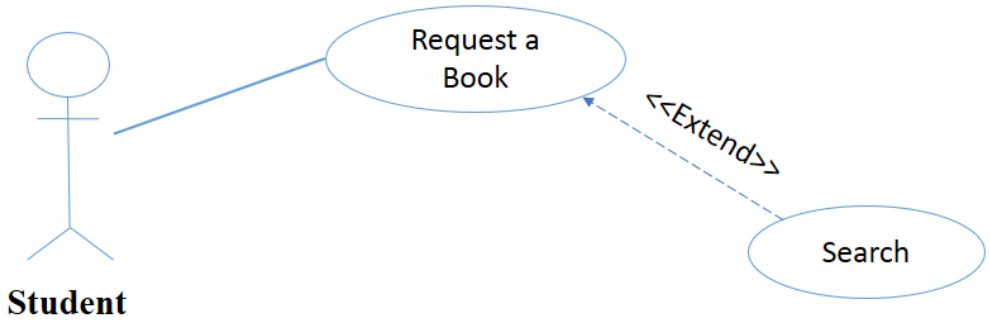


Figure 6.0.8 Extend Relationship

Generalization Relationship

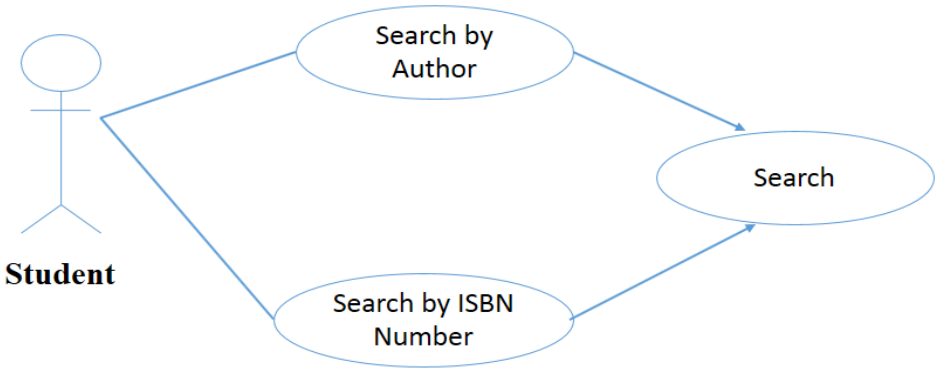


Figure 6.0.9 Generalization Relationship

How to Identify Actor

- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- What other systems use this system?
- Who gets information from this system?

- Who provides information to the system?

How to Identify Use Case?

- What functions will the actor want from the system?
- Does the system store information? What actors will create, read, update or delete this information?
- Does the system need to notify an actor about changes in the internal state?
- Are there any external events the system must know about? What actor informs the system of those events?

Use Case Diagrams RULES

- **An actor initiates a use case**, and an actor receives something of value from the use case.
- **Initiating actor is on the left of the use case, and receiving actor is on the right.**
- Actors name appears just below the actor.
- Name of the use case appears either inside the ellipse or just below it.
- An association lines connect an actor to the use case and represent the communication between actor and the use case.
- Actors are outside of the system and use cases are inside the system.
- Use a rectangle to represent the system boundary.

Exercise 01: Banking System

A Customer open a Savings Account with the help of Bank Employee. Customer can Deposit funds as well as Withdraw funds. Customer can Check the Account Balance and later customer can Close the Account if he/she wants. Customer can obtain the Loans with the help of Manager.

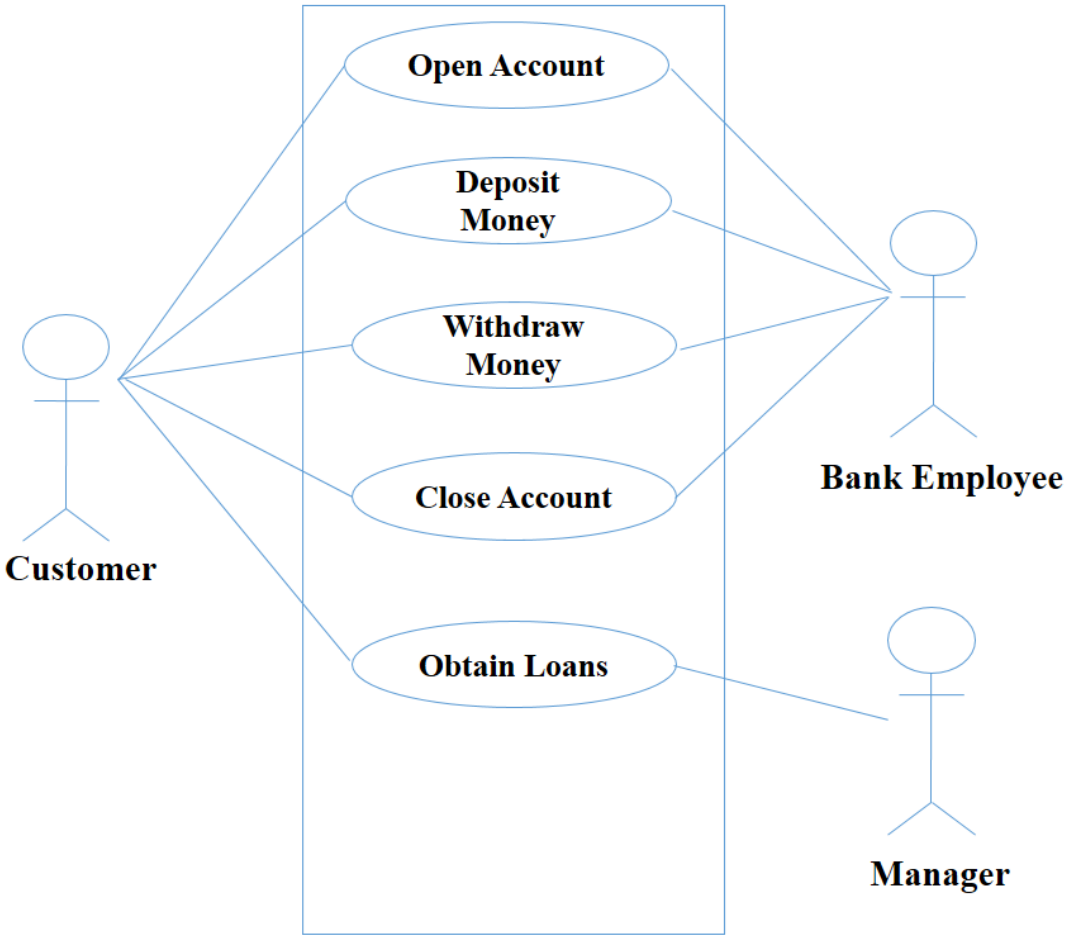


Figure 6.0.10 Use case diagram – Banking system example 1

Exercise 02: Banking System

There are three Customer Types: Online Customer, Walking Customer and NRFC Customer. Walking Customer open a Savings Account with the help of Bank Employee, But Online Customer and NRFC customer can open a savings account Online without Bank Employee.

Customer can Deposit funds as well as Withdraw funds. When deposits funds Bank calculate the Bonus if the amount is over 10,00 and if the customer's age over 45. When Depositing and Withdrawing the money Updated Balance is a mandatory function.

Customer can Check the Account Balance.

Customer can obtain the Loans with the help of Manager.

NRFC Customer Can Convert the Currency.

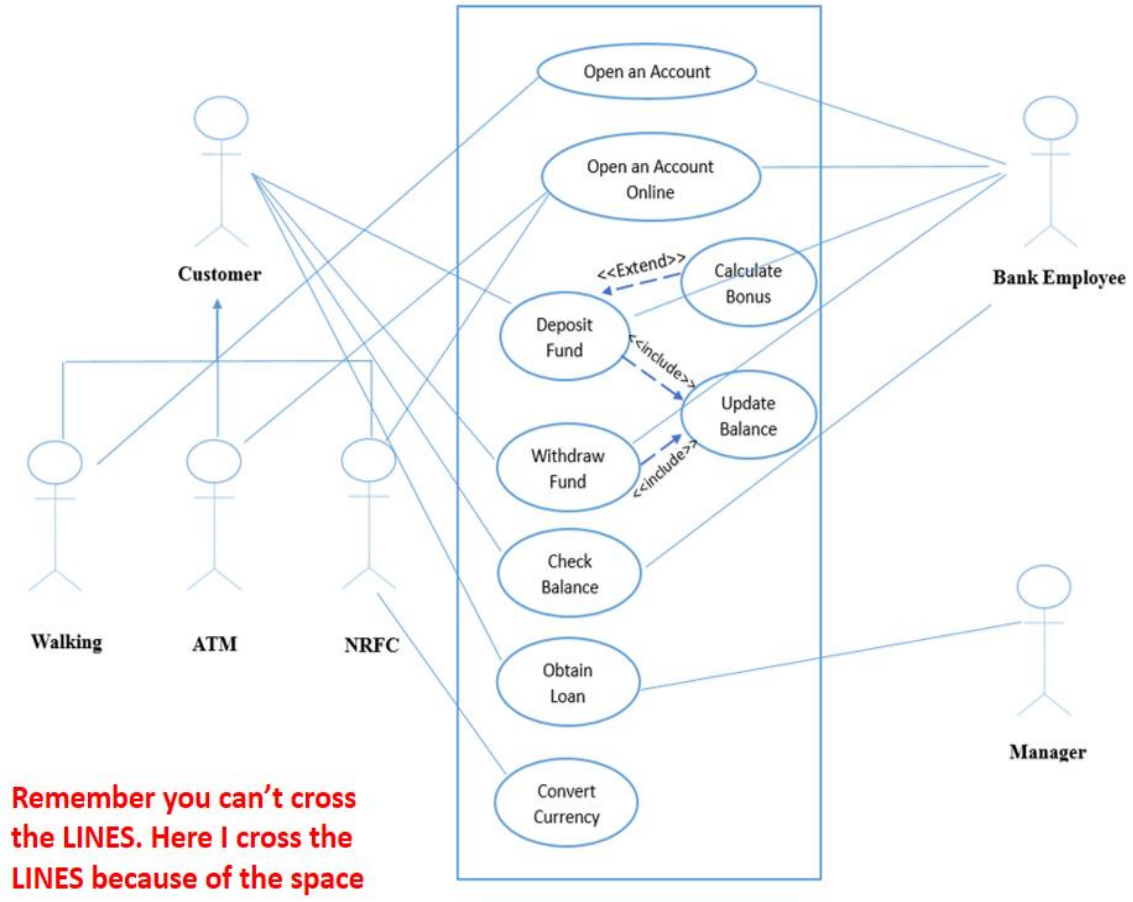


Figure 6.0.11 Use case diagram – Banking system example 2

Exercise 03: Student LMS System

The university wants to develop and automated system that would enable Program Office Staff to add the new Course details to the System. Also they can update and delete the course details. Before add the new course details staff account should be verified.

Lecturer can add modules to the course. Students can view all the details of all modules offered for a particular semester. Student can drop any module which he/she already registered and submit any assignment online. The system should also enable the lectures to check lecture schedules and access the submitted work of the students.

When submitting the Assignments students can view the Lecturer name if they needed.

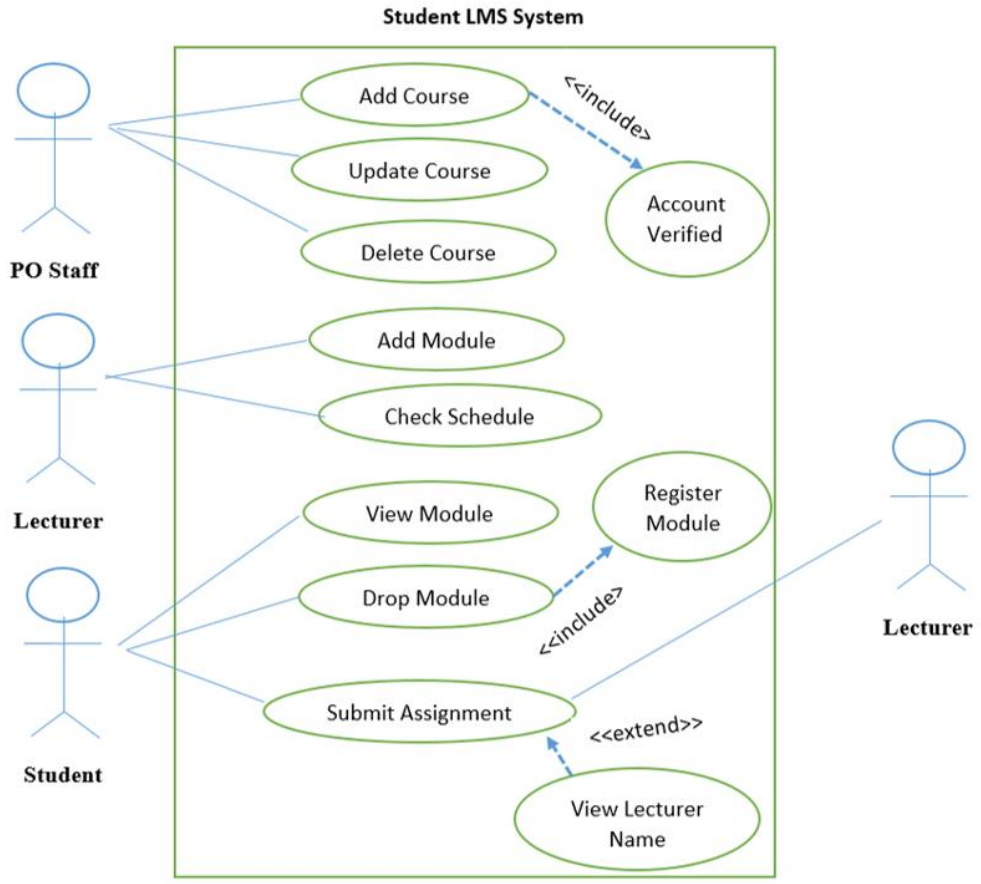


Figure 6.0.12 Use case diagram – Student LMS system

Exercise 04: Online Shopping System

Food City wants to develop an Online Shopping System where Customer can register first, then Customer Can Place his/her order. Before place the order customer **must check the availability of the item.** After placing the order customer can view the Order as well.

Customer Service of the Food City can view the order of the customer and update the order status. Also they can add, update, delete items from the system. When item is not available purchasing department create a purchase order and send it to the Supplier. Before that supplier should be register in the system. Once the items delivered, purchasing department maintain a GRN and update the inventory.

Customer can do the payments using credit cards or cheque. System automatically generates Monthly Sales Report and Daily Stock Report which can be view by only Branch Manager.

When viewing the reports Manager can change the options of the display report (Example: Monthly to Quarter, or Daily to Week)

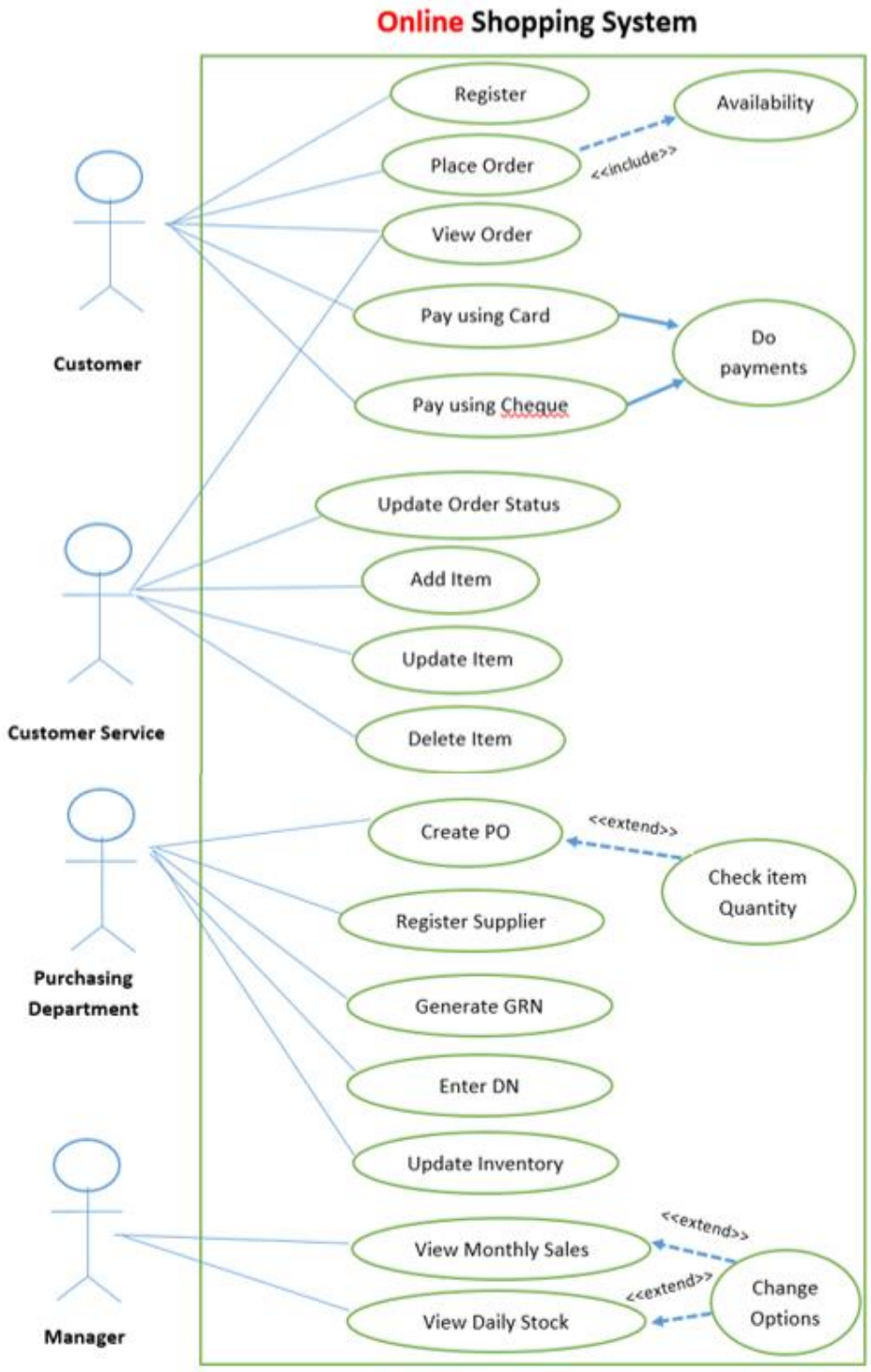


Figure 6.0.13 Use case diagram – Online shopping system

UML Diagrams Tools

Since it is very difficult to draw Use Case and other diagrams in word document you can use following web site to sign up for free UML designs.

You can use **Microsoft VISIO** tool instead of Word. Or else you can use following Online Tools for UML design.

https://www.lucidchart.com/

https://www.smartdraw.com/

Lesson 07 – Software Design Using Object Oriented Analysis (Class Diagrams)

Class Diagram

Class Diagram describes the structure of a system by showing the system's

- Classes
- Attributes of the Classes
- Operations (or methods) of the Classes
- Relationships among Objects

A Class is a **Blueprint for an Object**. We can't talk about Class without talking about the Object. Objects have states and behaviors. Each Object was built from the same set of blueprints and therefore contains the same components (properties and methods).

The standard meaning is that an object is an instance of a class and object.

Example - A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

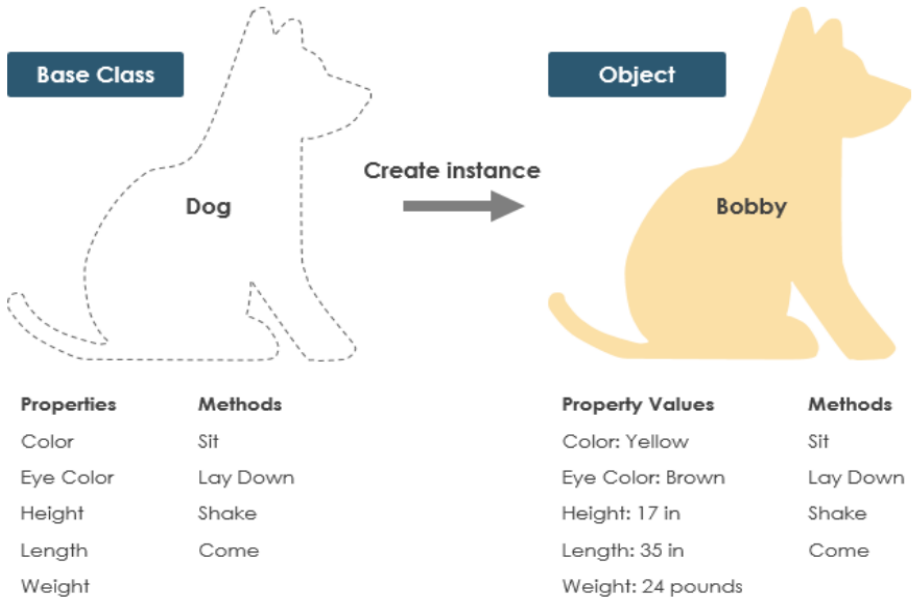


Figure 7.0.1 What is a Class?

Class Diagram Notation and Symbol

- A Class represent a concept which Encapsulates state (attributes) and behavior (methods).
- A Class Notation consists of three parts:

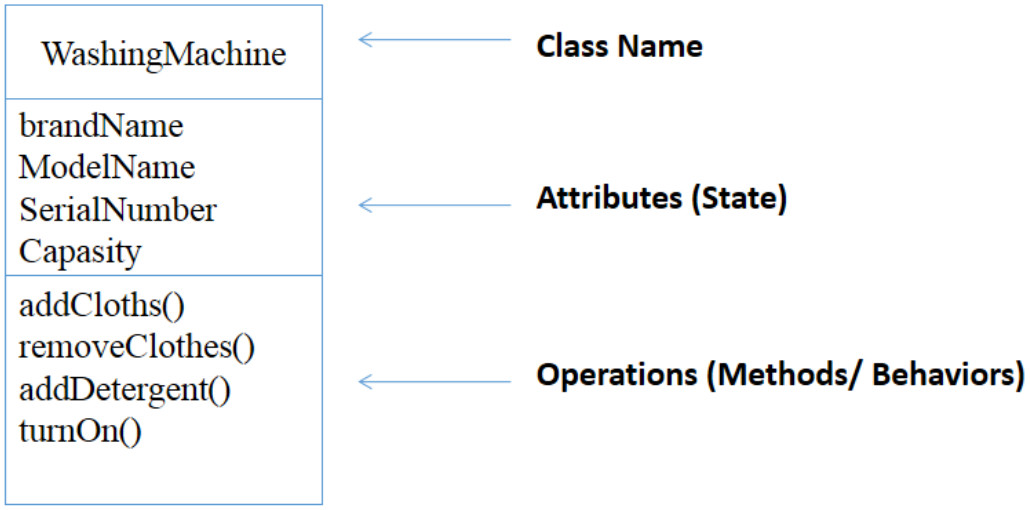


Figure 7.0.2 Class Notation

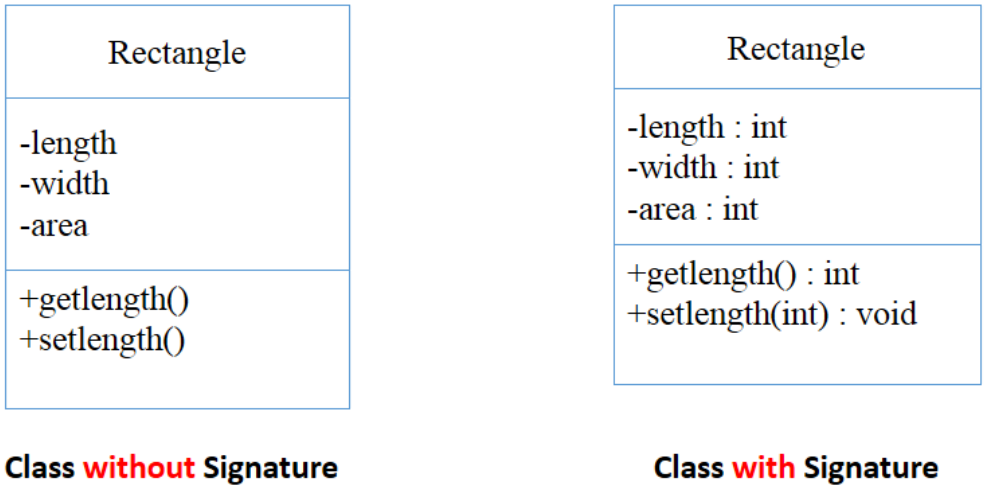


Figure 7.0.3 Class with & without signature

Class Name:

- The name of the class appears in the first partition.

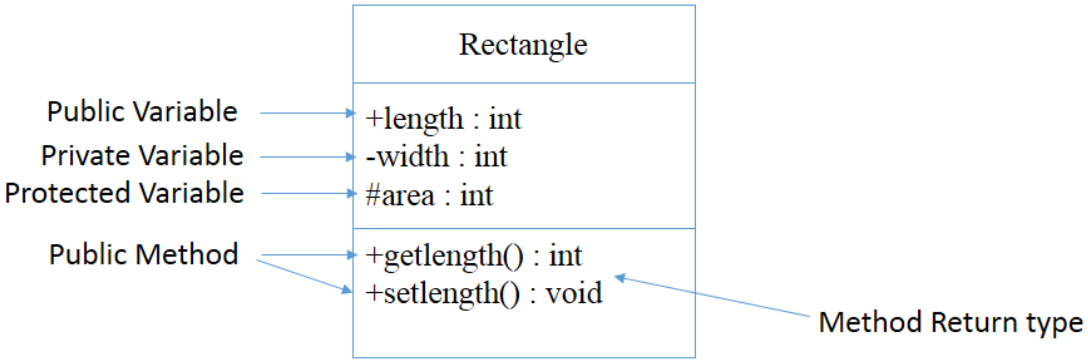
Class Attributes:

- Attributes are shown in the second partition.
- The attribute type is shown after the colon.
- Attributes map onto member variables (data members) in code.

Class Operations (Methods):

- Operations are shown in the third partition. They are services the class provides.
- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters is shown after the colon following the parameter name. Operations map onto class methods in code.

Class Diagram Visibility



The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

+ denotes public attributes or operations

- denotes private attributes or operations

 # denotes protected attributes or operations

Class Diagram Visibility with all the options

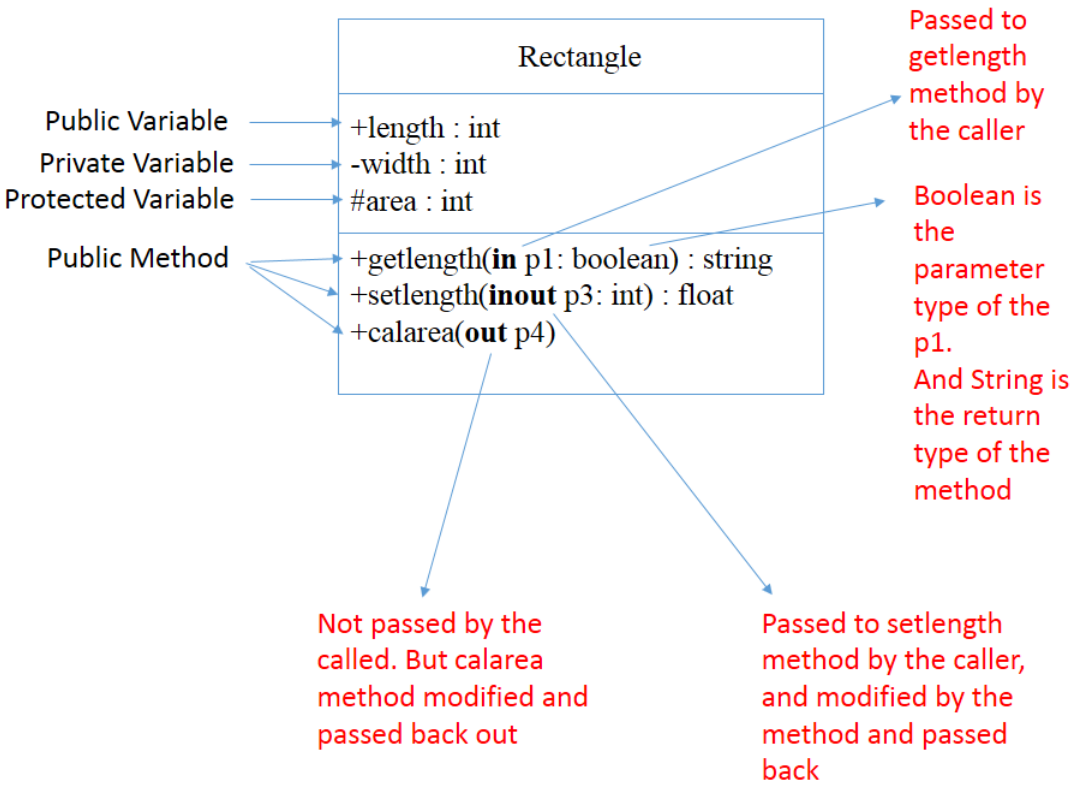


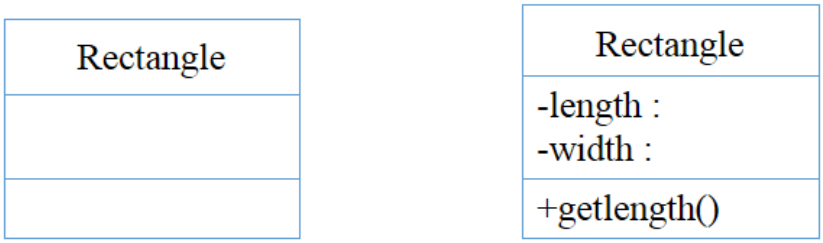
Figure 7.0.4 Class Diagram Visibility with all the options

Perspectives of Class Diagram

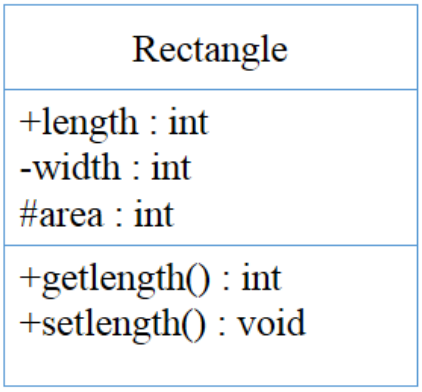
A diagram can be interpreted from various perspectives:

- **Conceptual**: Represents the concepts in the domain

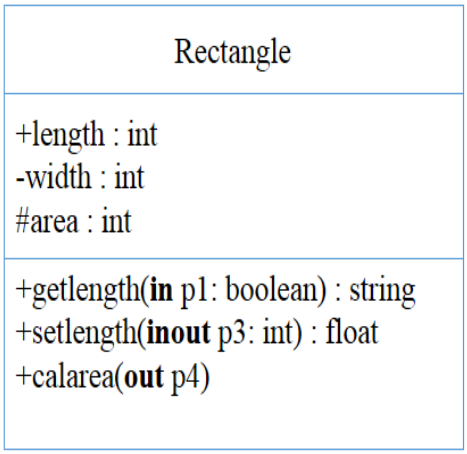
The class name is the only mandatory information.



- **Specification**: Focus is on the interfaces of Abstract Data Type (ADTs) in the software.



- **Implementation**: Describes how classes will implement their interfaces.



Relationships between Classes

A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:

Inheritance (or Generalization)

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class.

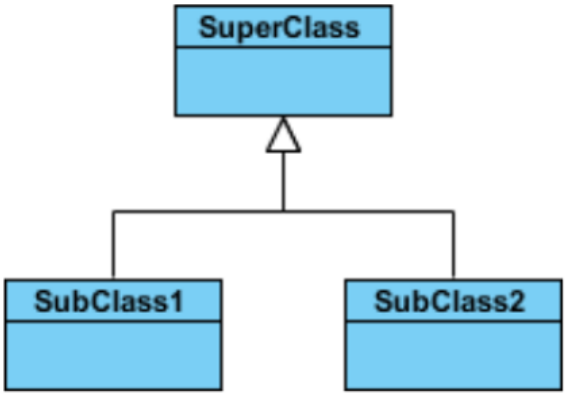
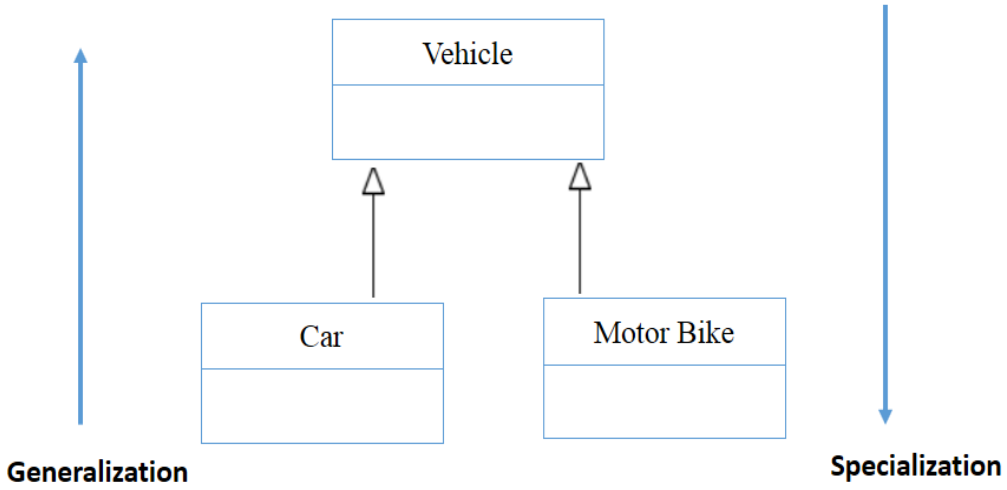


Figure 7.0.5 Relationship between classes - Inheritance

Example for Inheritance (or Generalization):



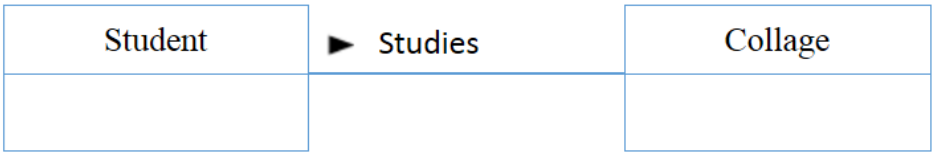
Association

- Associations are relationships between classes in a UML Class Diagram. They are represented by a solid line between classes.
- Associations are typically named using a verb or verb phrase which reflects the real world problem domain.

Simple Association

- A structural link between two peer classes.
- There is an association between Class1 and Class2.

Example for Simple Association:



Names of relationships are written in the middle of the association line. They often have a **small arrowhead to show the direction** in which direction to read the relationship.

Multiplicity

- How many objects of each class take part in the relationships?

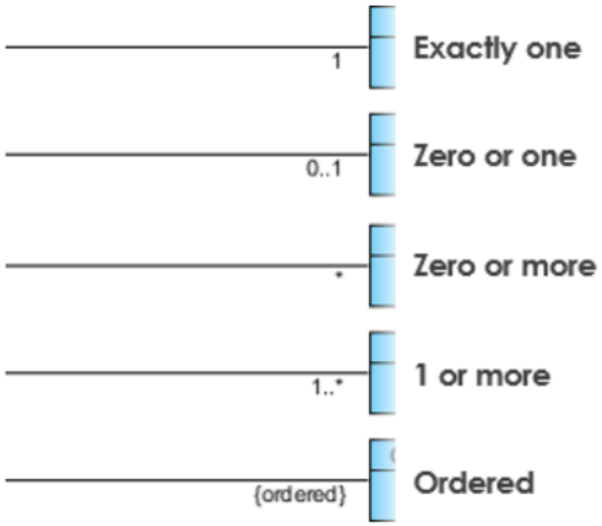
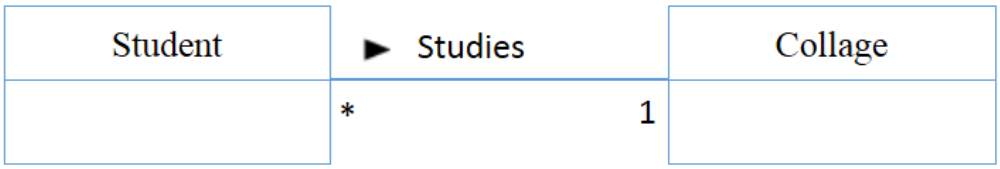


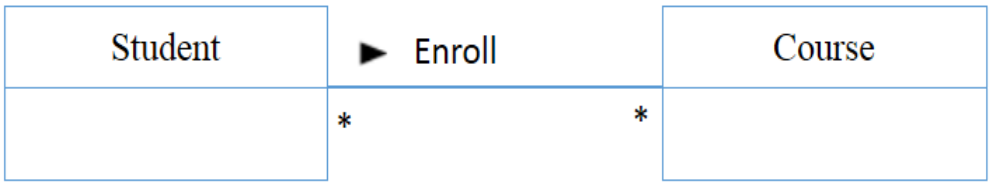
Figure 7.0.6 Multiplicity

Examples for Multiplicity:

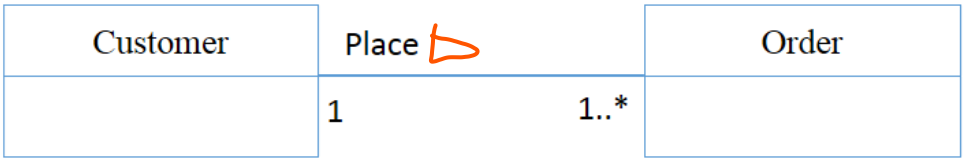
Ex: The college can have multiple students.



Ex: A Student can take many Courses and many Students can be enrolled in one Course.

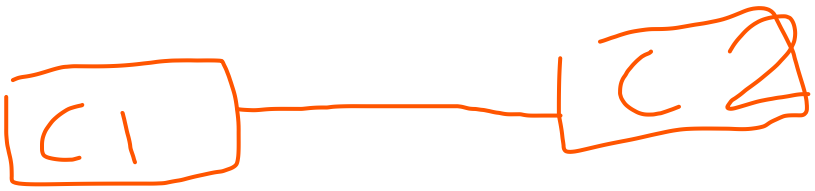


Ex: One Customer can place one or more orders.



Aggregation

- **A special type of Association.**
- It represents a "part of" relationship. (“Consists-of” hierarchy)
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.



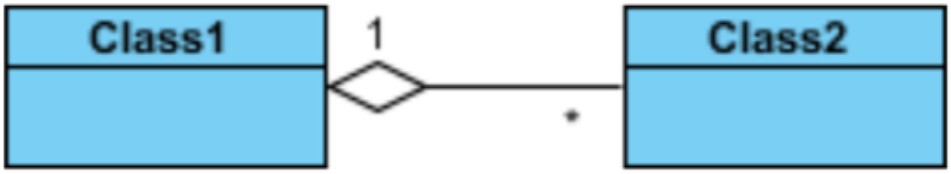
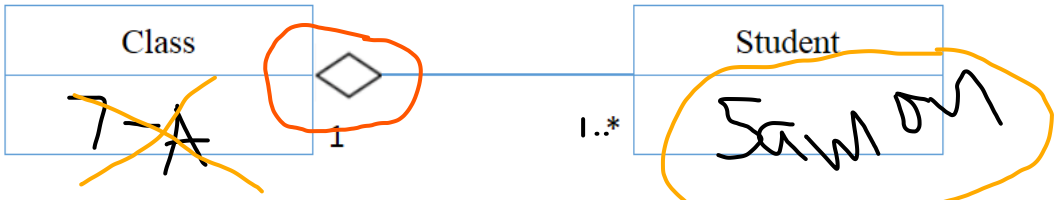


Figure 7.0.7 Aggregation

Example for Aggregation:



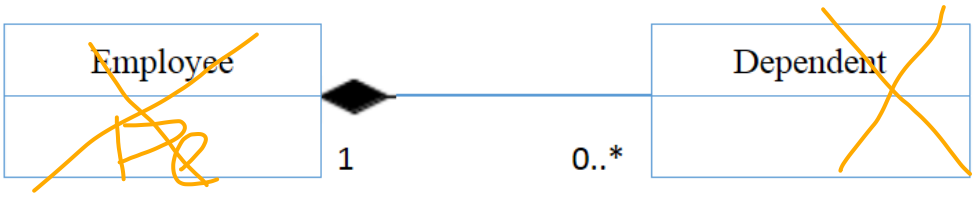
For example, the Class is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the **Class will remain even if the student is not available.**

Composition:

- **A special type of Aggregation** where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite.

Figure 7.0.8 Composition

Example for Composition:



For example, the Employee have zero or many dependents. This relationship is composition relationship **because without the Employee class Dependent class cannot be exit.**

Dependency:

- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2.
- A dashed line with an open arrow.

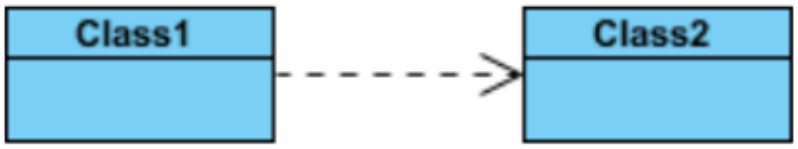


Figure 7.0.9 Dependency

Example for Dependency:

Here Course Schedule class is depending on the Course. If something changed in the Course Automatically Course Schedule can be changed.

The figure below shows the three types of association connectors: **Association, Aggregation, and Composition.**

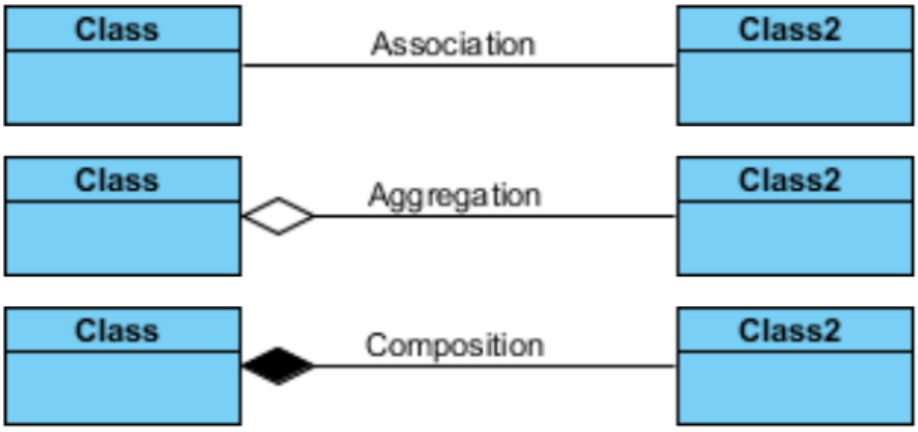


Figure 7.0.10 Association, Aggregation, and Composition

The figure below shows Generalization (Inheritance).

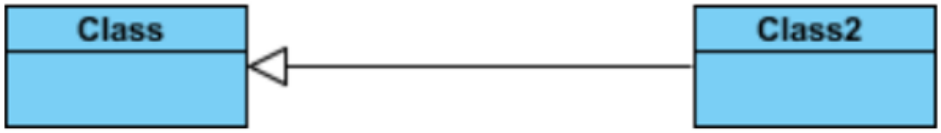


Figure 7.0.11 Generalization (Inheritance)

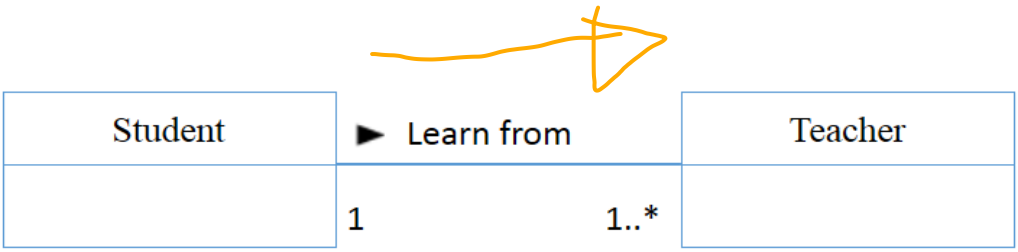
Exercise

What kind of a relationship can be exits in following Classes?

1. A Single Student can associate with Multiple teachers:
2. Every Instructor has one or more Students:
3. Customer has one or more Accounts:

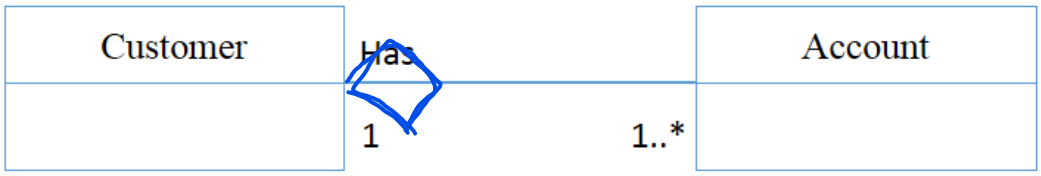
Answers

1. A single student can associate with Multiple teachers:



2. Every Instructor has one or more Students:

3. Customer has one or more Accounts:



Aggregation vs Composition

- **Aggregation** and **Composition** are subsets of association meaning they are **specific cases of association**.

- In both aggregation and composition object of one class "owns" object of another class. But there is a subtle difference:

- **Aggregation** implies a relationship where the **child can exist independently of the parent.** Example: Class (parent) and Student (Child). Delete the Class and the Students still exist.

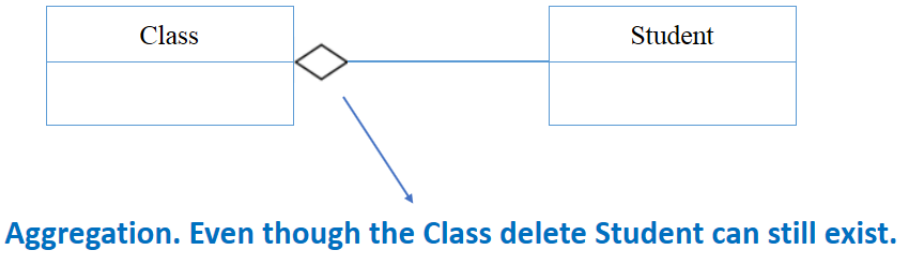


Figure 7.0.12 Aggregation

- **Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (Parent) and Room (Child). Rooms don't exist separate to a House.

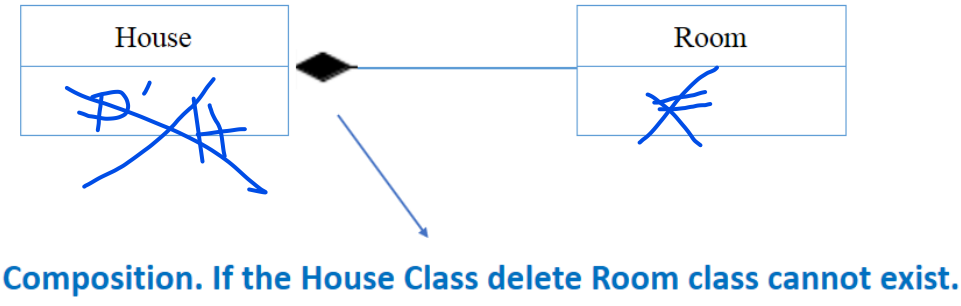
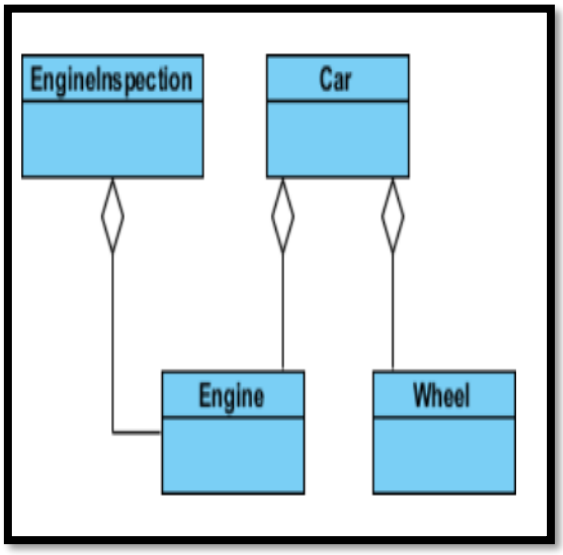
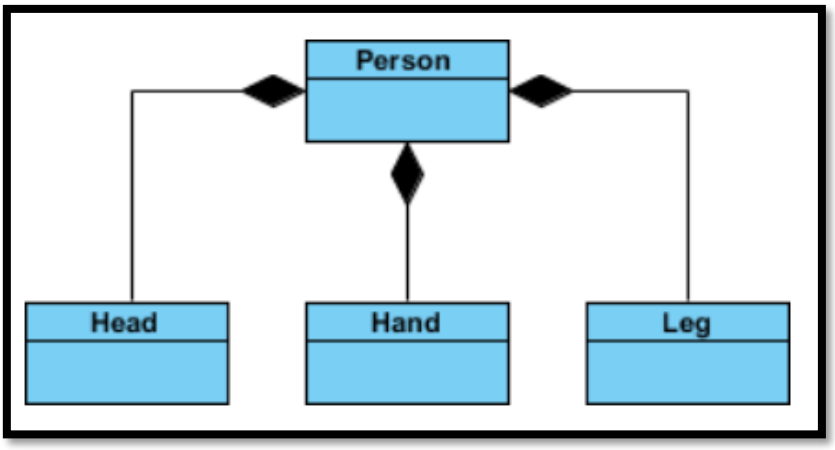


Figure 7.0.13 Composition

Aggregation



Composition



Exercise 1: ATM System

Bank has different ATM machines in different locations managed by different branches. Each Bank Branch has a Code and an Address. Using the ATM, Customers can do the Transactions.

Customer has an id, name, address, card number and a pin number. Before do any transactions password should be verified. Customers have Accounts. Account has a number and balance. Deposit and Withdraw can be done through Account.

Through the Account, ATM Transactions can be maintained. For each ATM Transaction date, transaction id, amount, post balance recorded. Savings Accounts and Current Accounts are the two types of Bank Accounts. Both accounts have an account no and balance. When creating a Current Account Bank always checks the Savings in the Saving Account.

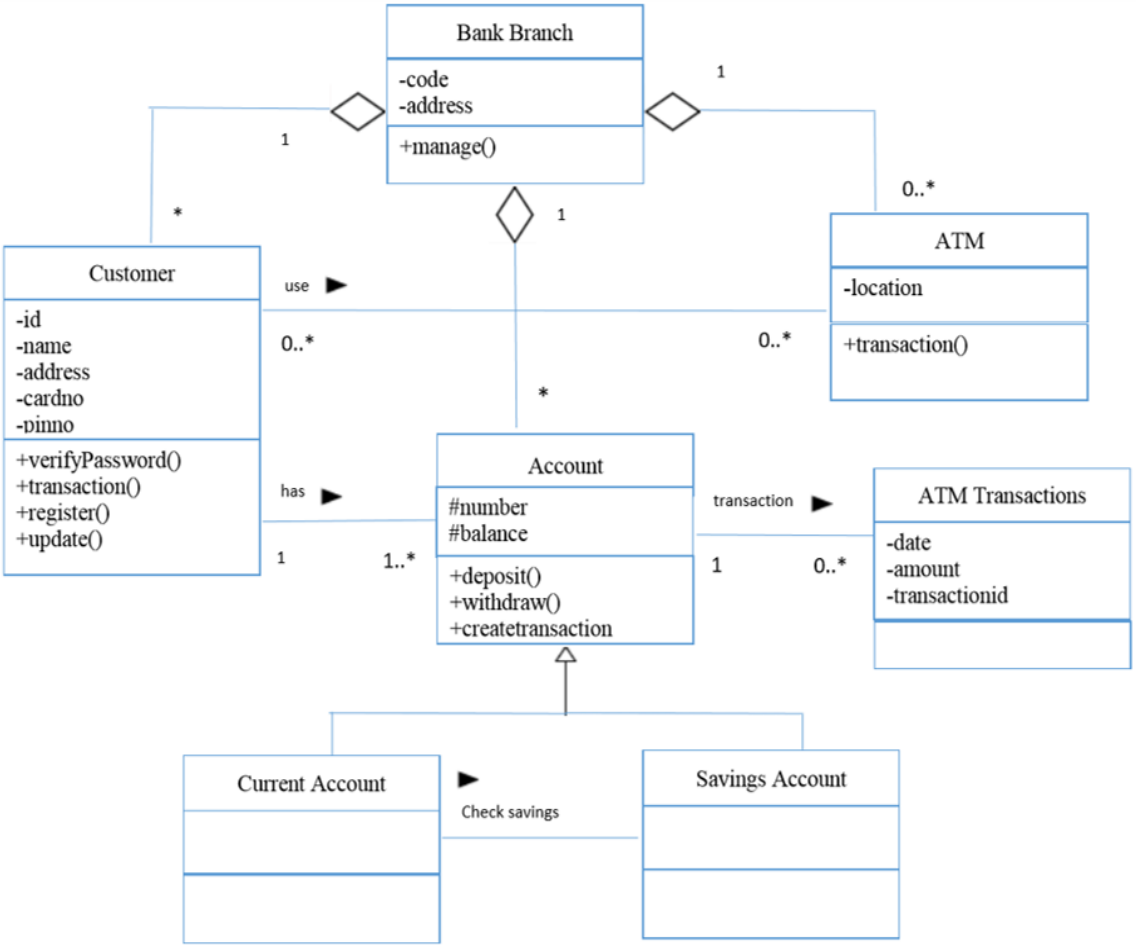


Figure 7.0.14 ATM System – Exercise 1

Exercise 2: Student Management System

At NIBM there are different departments which offer different courses to the Students. Each department and course has its own id and name. A Student can register under a course which offer by specific department and student's id, name, address and NIC should maintain. Students are categorized as Part Time and Full Time students where Part Time student need job experience to follow part time course. In each department there are Lecturers who conduct lectures for different courses. Lecturer's Id, Name, qualification need to be record.

Lecturer create course schedule, send to students and course schedule is depends on the course and the availability of the Hall. Each hall has a hall number and a capacity. Lecturers are responsible for the courses. You have freedom to add required methods (operations) for each Classes by analyzing the question. Ex: In Student Class following methods can be added. Add_Student(), Update_Student(), Inactive_Student(), Transfer_Student etc.

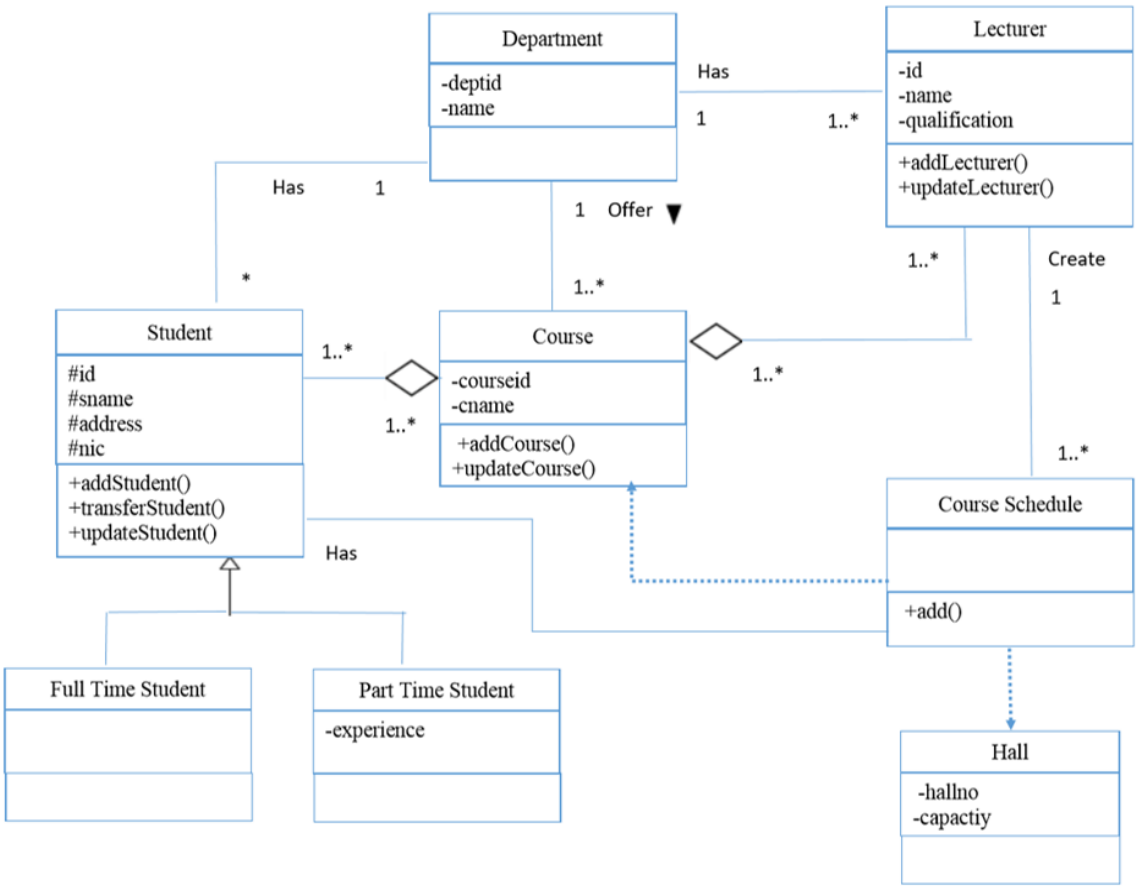


Figure 7.0.15 Student Management System – Exercise 2

Exercise 3: Online Shopping System

Food City wants to develop an Online Shopping System where Customer can register using Customer Name and Contact Number, then Customer Can Place his/her order. The system should record Order Id, Item details and Delivery Address.

The Order can view and update the order status by the Customer Service. Also they can add, update, delete items from the system. System should register suppliers and supplier will get purchase order whenever food city wants to order items from the supplier. Once the items delivered Good Receive Note created. Once the customer does the payments invoice should be created. Also Customer can return the purchased goods and Food City can return the goods back to the Supplier.

You have freedom to add required methods (operations) for each Classes by analyzing the question. Ex: In Supplier Class following methods can be added. Add_Supplier(), Update_Supplier(), Inactive_Supplier().

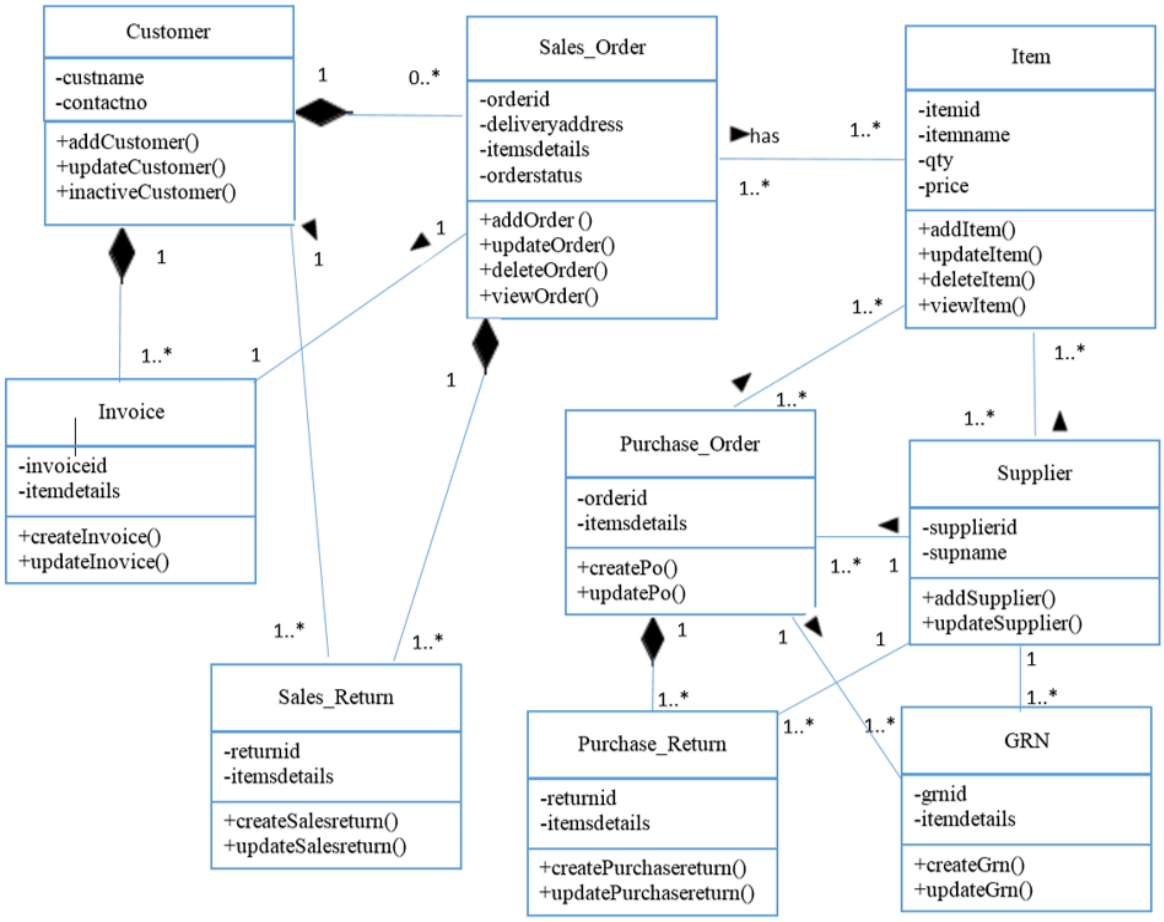


Figure 7.0.16 Online Shopping System – Exercise 3

Lesson 08 – Software Design Using Object Oriented Analysis (Sequence Diagrams)

Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.

Sequence Diagrams captures: high-level interactions between user of the system and the system, between the system and other systems, or between subsystems.

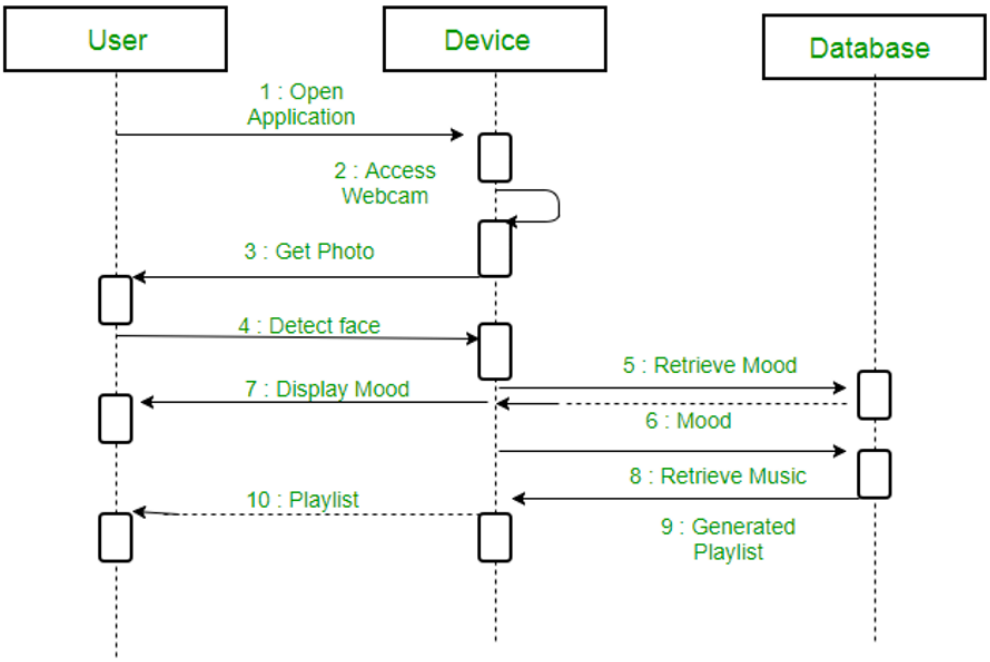


Figure 8.0.1 Sequence Diagram

Sequence Diagram Notation and Symbol

- **Object**

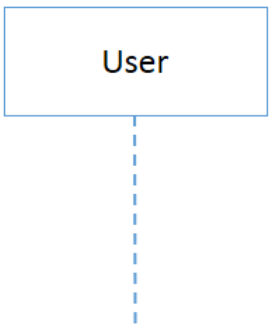
Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system.

- **Actor**

Shows entities that interact with or are external to the system.

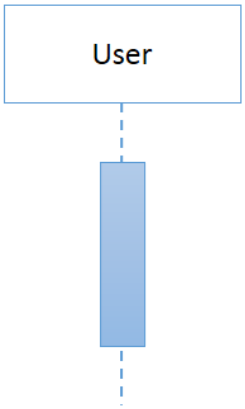
- **Lifeline**

Lifelines are vertical dashed lines that indicate the object's presence over time. Lifelines may begin with an **Object or an Actor** symbol.



- **Activation**

Represents the time needed for an object to complete a task. **An activation is represented by a thin rectangle on a lifeline**. It represents the period during which an element is performing an operation.



Messages Types

- **Call Message (Synchronous Message):**

Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues.

- **Asynchronous Message:**

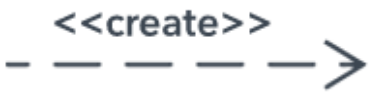
Represented by a solid line with a lined arrowhead. Asynchronous messages don't require a response before the sender continues.

- **Reply Message (Return):**

Represented by a dashed line with a lined arrowhead, these messages are replies to calls.

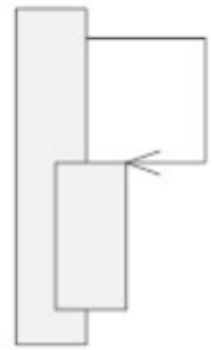
- **Asynchronous create message:**

Represented by a dashed line with a lined arrowhead. This message creates a new object.



- **Self-Message:**

A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.



- **Delete Message:**

This is a message that destroys an object. It can be shown by an arrow with an x at the end.

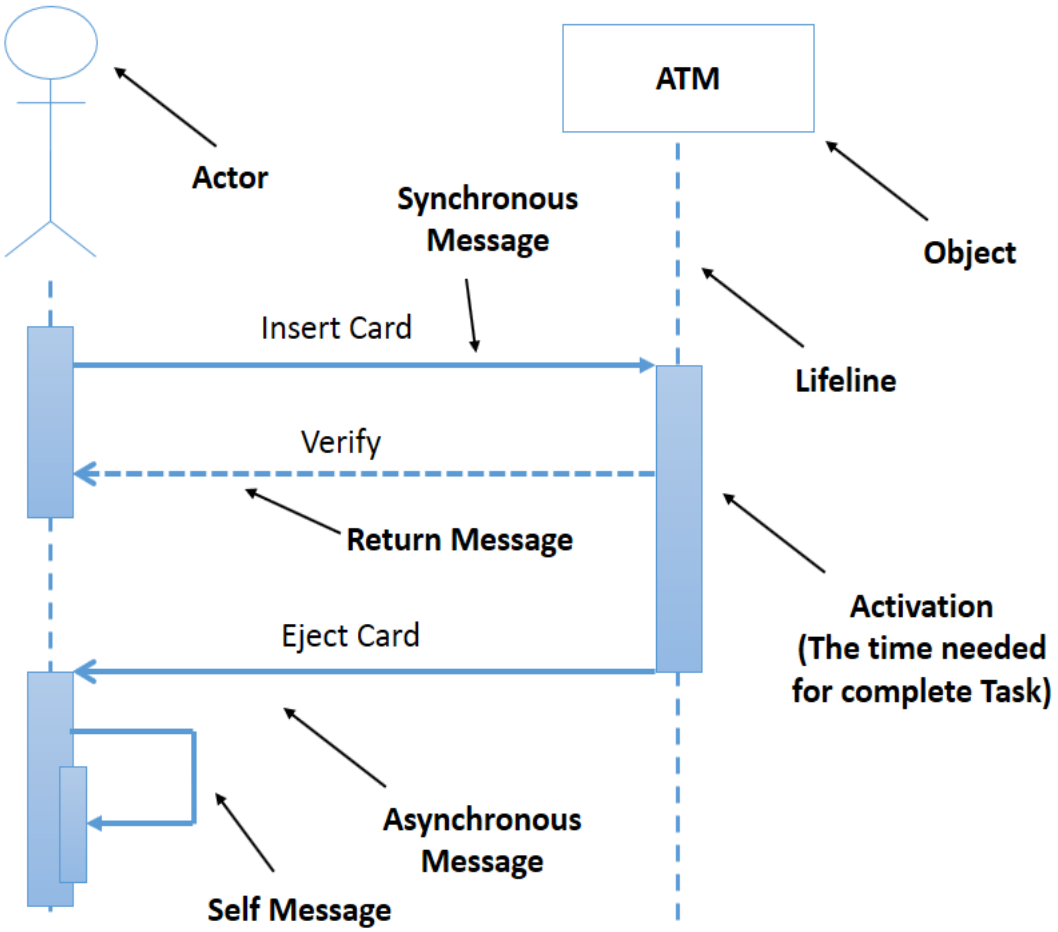


Figure 8.0.2 Sequence Diagram - Example

Exercise 01: ATM System

Customer enter the Account No into **ATM** machine. The Account No should verify from the **Database** through a **Controller**, also Customer should wait until the Account No verify. All the time ATM sends message to the Database through the Controller. After the account no verify ATM display Welcome Message. ATM request amount to be deposit and customer can enter the deposit the amount.

Deposit amount should save in the database and same time it updates the customer account. ATM return the feedback back to the Customer.

Customer enter the ATM Card into **ATM** machine. The Pin should verify from the **Database** through a **Controller,** also Customer should wait until the pin verify. After the pin verify ATM display Main Menu. ATM request amount to be withdraw and customer can enter the withdraw amount. Whether the amount is sufficient or insufficient check against the database. Based on the Database feedback ATM return a message back to the Customer.

Finally, customer can eject the card from the ATM.

Exercise 01: ATM System - Use Case Diagram

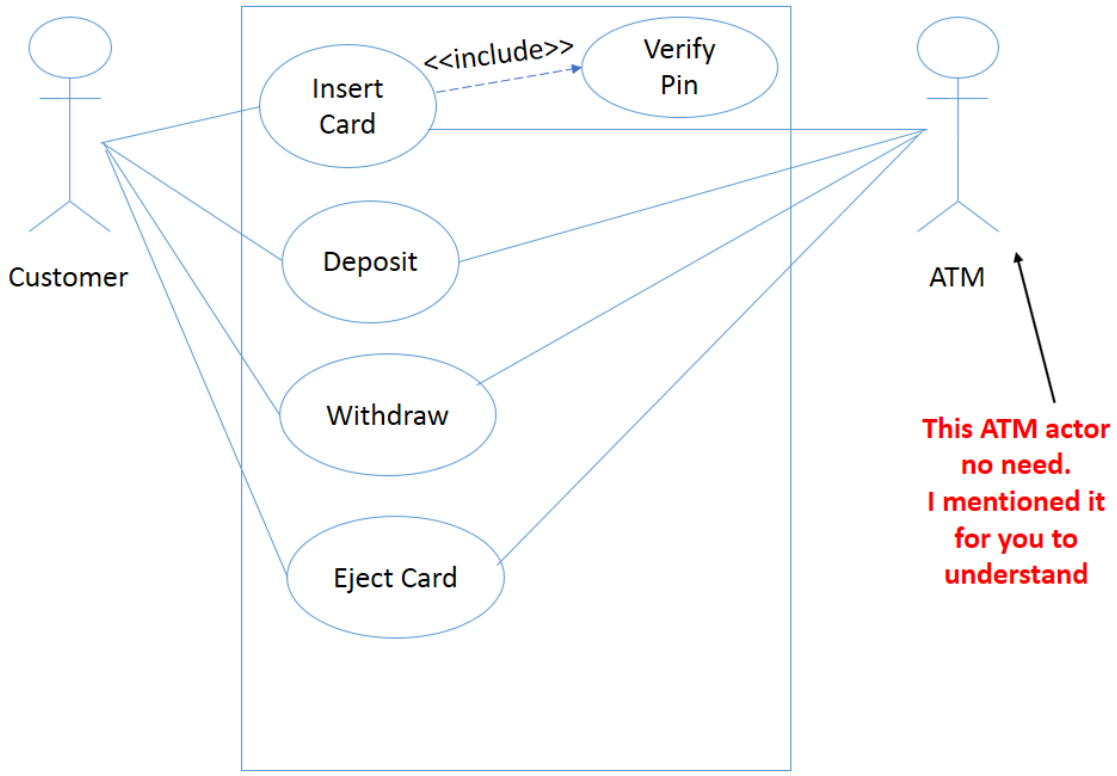


Figure 8.0.3 Use Case Diagram - ATM System

Exercise 01 - Sequence Diagram for Deposit Fund

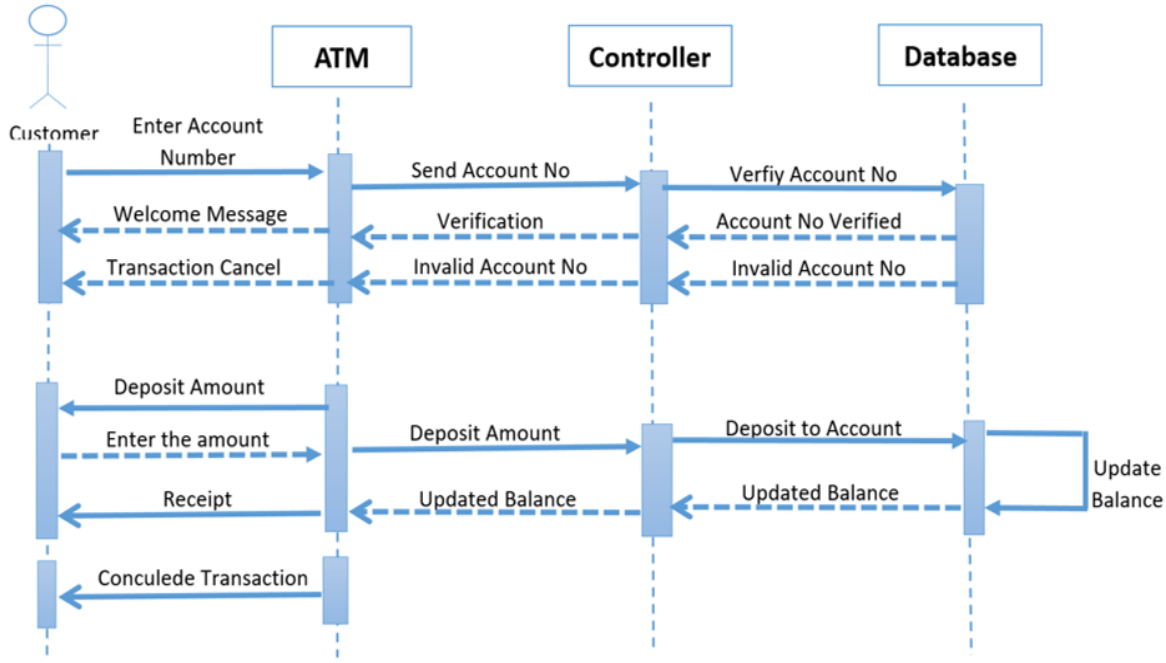


Figure 8.0.4 Sequence Diagram for Deposit Fund

Sequence Diagram for Withdraw Fund

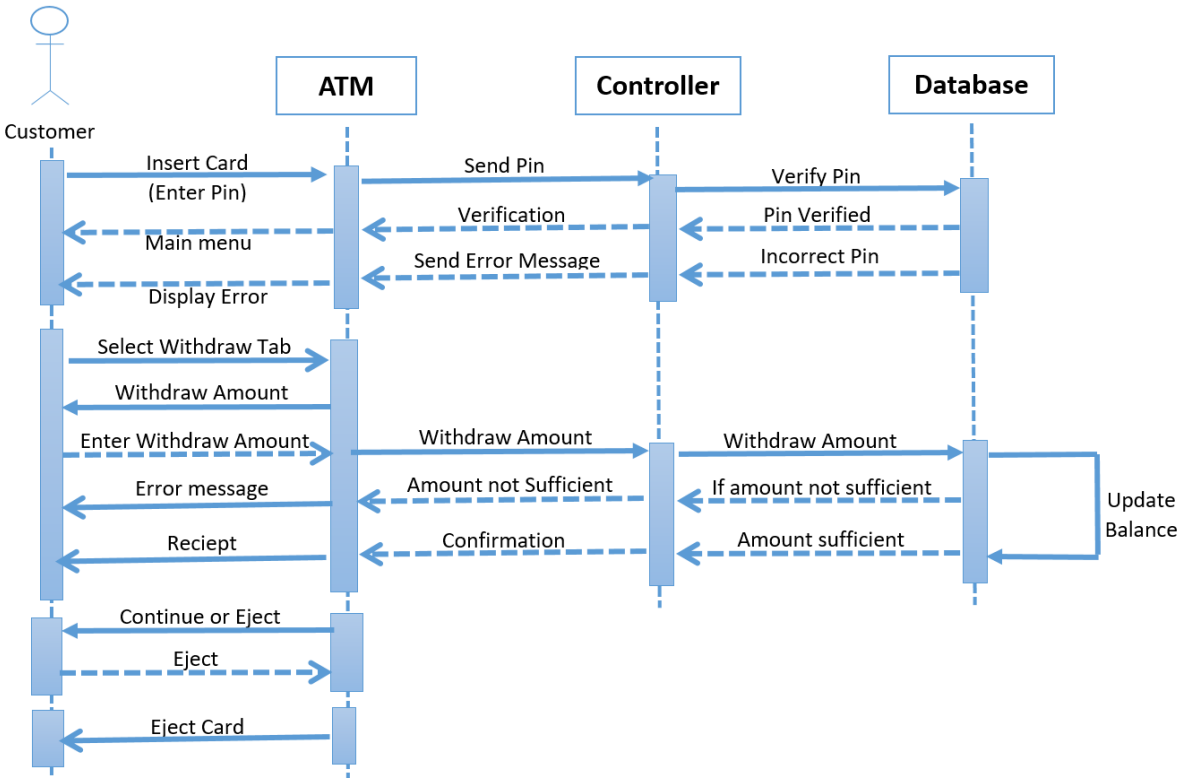


Figure 8.0.5 Sequence Diagram for Withdraw Fund

Exercise 02: Online Shopping System

Customer can sign up online by giving his/her detail, then customer will get user name and password through an email. Customer can login and can Place his/her order. Before place the order availability of the item should be checked.

Customer Service of the Food City can view the order of the customer and update the order status. Also they can add, update, delete items from the system. When item is not available purchasing department create a purchase order and send it to the Supplier.

Once the items delivered, purchasing department maintain a GRN and update the inventory.

Customer can do the payments using credit cards or cheque. System automatically generates Monthly Sales Report and Daily Stock Report which can be view by only Branch Manager. **Remember you must Design sequence diagrams for every system function.**

Sequence Diagram for Sign Up Customer

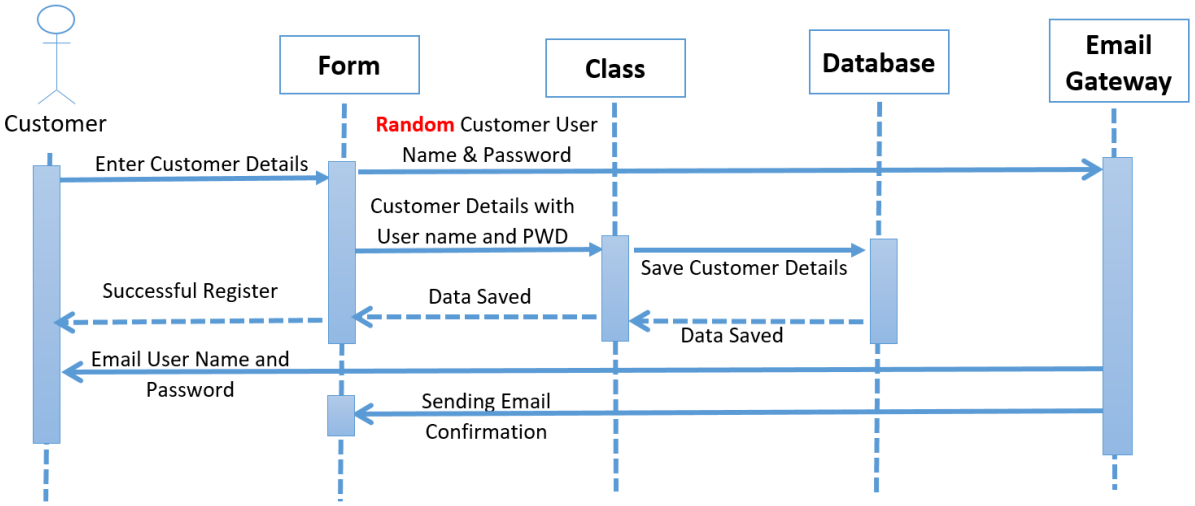


Figure 8.0.6 Sequence Diagram for Sign Up Customer

Sequence Diagram for Login and Place Order

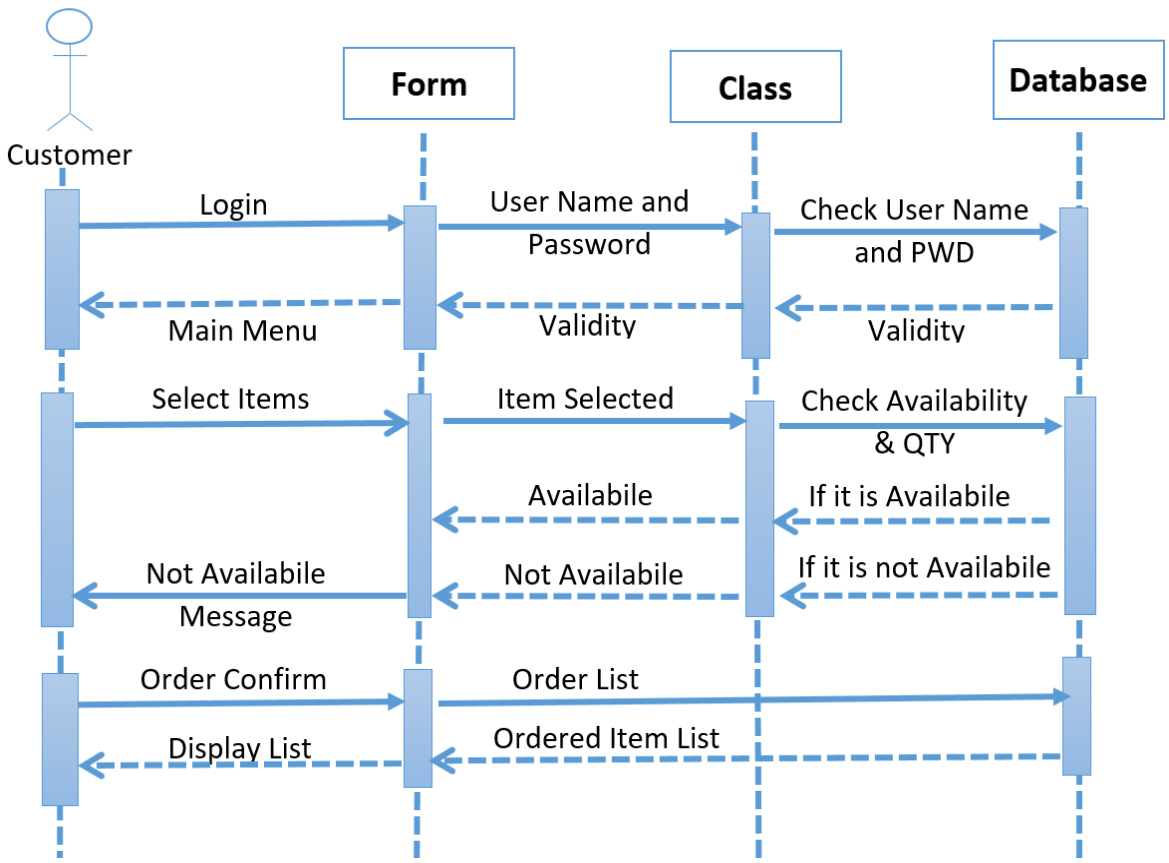


Figure 8.0.7 Sequence Diagram for Login and Place Order

Lesson 09 – Database Design Using ER, Schema and File Design

Database Design Process

For designing an efficient, useful database you have to follow following steps:

1. Requirements analysis, or identifying the purpose of your database
2. Design Entity Relationship Diagram (ER)
3. Map the Entity Relationship Diagram to Relational Schema
4. Normalize the Schema up to Boyce-Codd Normalize Form
5. Design the tables with Field Size, Data Types, PK and FK
6. Physical design the Database and Tables in DBMS

Step 01: Requirements analysis, or identifying the purpose of your database

- Gather the Requirements from the Customer and Users.
- You have to focus on what kind of data they need to store, what kind of security they need etc.
- After gather the Requirements you need to identity what are the Entitles, Attributes and Relationships between Entities.
- Example: Suppose you have to develop a system for Hospital. Therefore, you have to identify Entities like Doctor, Patient, Ward, Treatment etc. Then the Relationships between above Entities.

Step 02: Design the Entity Relationship Diagram

- Based on the requirement design the ER or EER diagram

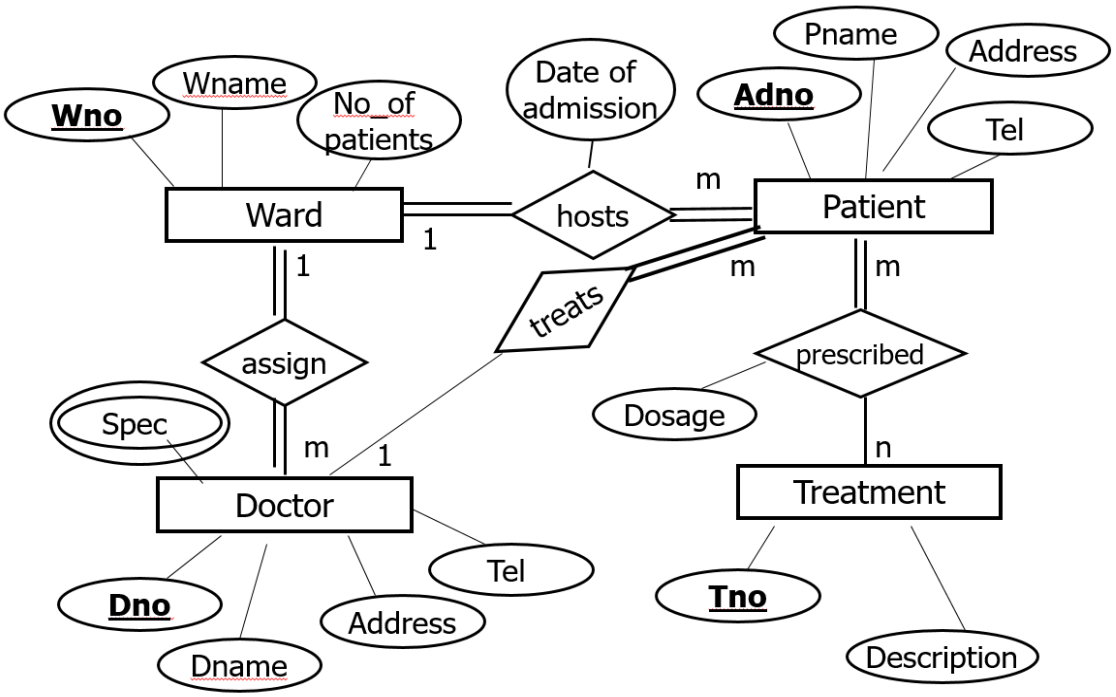


Figure 9.0.1 Entity Relationship Diagram

Step 03: Map the ER Diagram to Relational Schema.

ER Diagram map it to a Relational Schema using 7 Steps

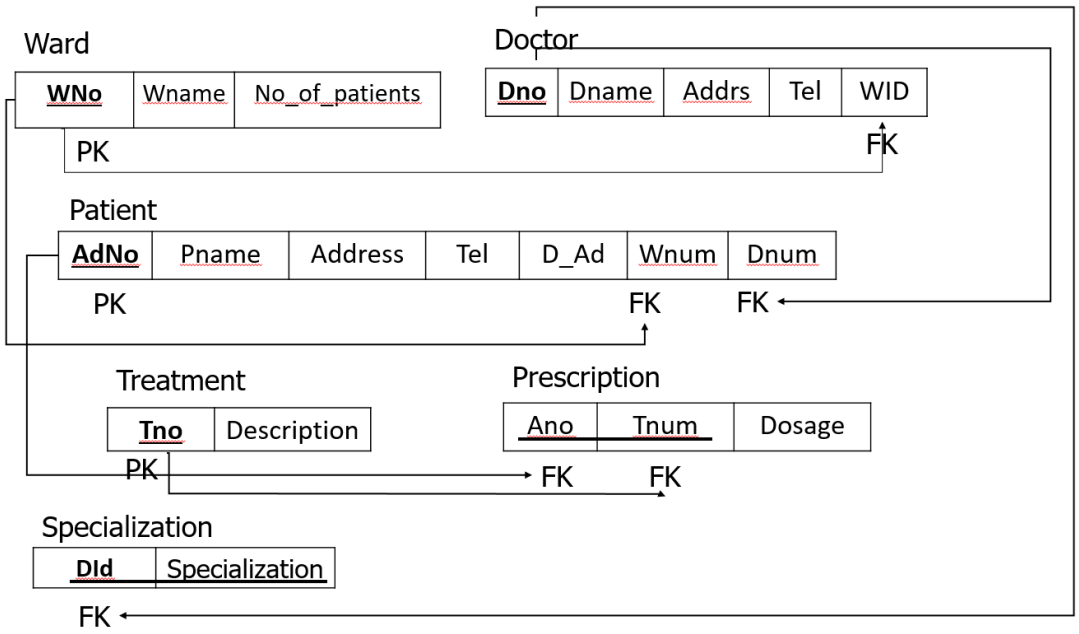


Figure 9.0.2 Relational Schema

Step 04: Normalize the Relational Schema to make sure there is no unsatisfactory relationships.

- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form

Step 05: Design the tables with Field Size, Data Types, PK and FK

Analyze the Filed Size and assign appropriate Data Types for each Field. Mentioned Primary Key and Foreign Key. Remember you must use the same name as mentioned in the Schema.

File Design for Hospital System

Table Name: Ward ← **Table Name**

Primary Key: WNo ← **Primary Key**

Filed Name	Data Type	Data Size
WNo	varchar	3
Wname	varchar	10
No_of_patients	int	3

← **Design the File using Filed Name, appropriate data type and data size**

Table Name: Doctor

Primary Key: Dno

Foreign Key: WID

Filed Name	Data Type	Data Size
Dno	varchar	3
Dname	varchar	10
Address	varchar	20
Tel	int	10
WID	varchar	3

← **Remember based on the DBMS software some times data size no needed**

Table Name: Patient

Primary Key: AdNo

Foreign Key: Wnum, Dnum

Filed Name	Data Type	Data Size
AdNo	varchar	3
Pname	varchar	10
Address	varchar	20
Tel	int	10
D_Ad	Date	
Wnum	varchar	3
Dnum	varchar	3

Table Name: Treatment

Primary Key: Tno

Filed Name	Data Type	Data Size
Tno	varchar	3
Description	varchar	20

Table Name: Prescription

Primary Key: Ano, Tnum

Foreign Key: Ano, Tnum

Filed Name	Data Type	Data Size
Ano	varchar	3
Tnum	varchar	3
Dosage	varchar	10

Table Name: Specialization

Primary Key: DId, Specialization

Foreign Key: DId, Specialization

Filed Name	Data Type	Data Size
DId	varchar	3
Specialization	varchar	15

Figure 9.0.3 File Design for Hospital System

Step 06: Create Physical Database and Tables using SQL queries in DBMS.

The Output will look like this inside the DBMS.

Ward

Wno	Wname	No_Of_Patients
W1	Heart	45
W2	Brain	55
W3	Neuro	75
W4		

Patient

Adno	Pname	Address	Tel	Wardno	Date_Ad	DID
P1	Gayan	Galle	077	W1	2012-01	D1
P2	Kasun			W2	2012-02	D2
P3	Wasana			W1	2012-01	D2
P4	Amali			W2	2012-01	D3

Doctor

Dno	Dname	Address	Tel	WID
D1	Perera	Col07	077	W1
D2	Kamal			W1
D3	Fernando			W2
D4	Kamala			W3

Treatment

Tno	Description
T1	Pandol
T2	Syrup
T3	Injection
T4	Anti

Prescribe

AdID	TID	Dosage
P1	T1	15
P1	T3	2
P2	T3	3
P3	T1	20

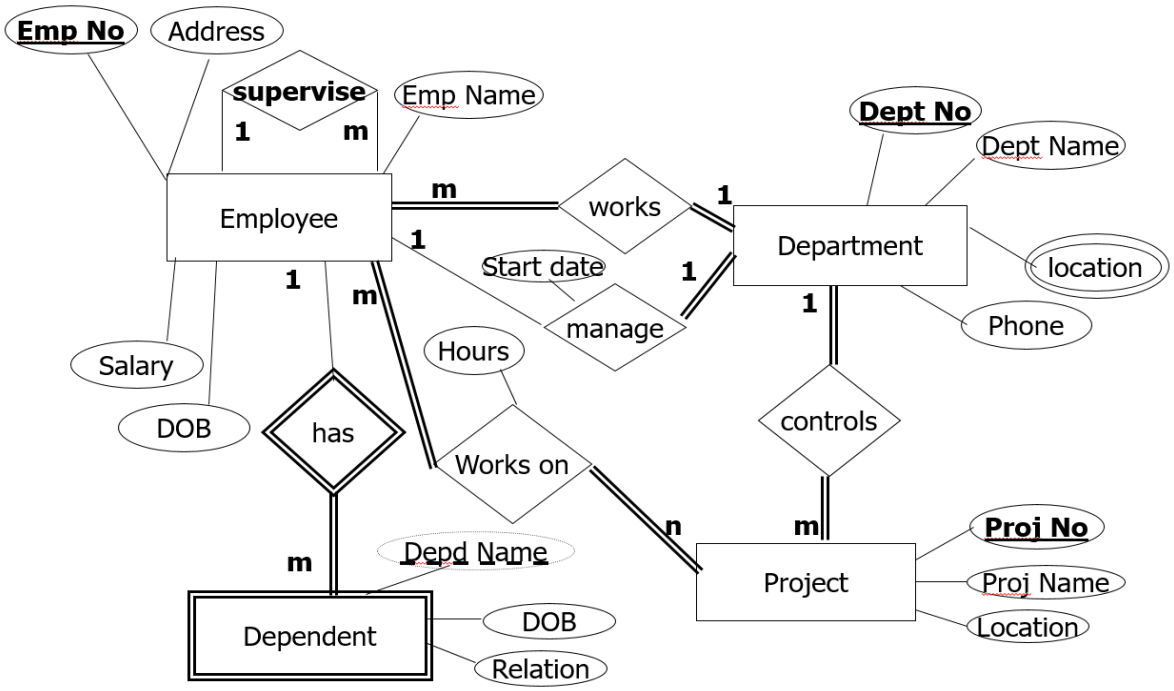
Doctor_Specilization

Docno	Spec
D1	Heart
D1	Neuro
D2	Heart
D3	Heart

Figure 9.0.4 Physical Database and Tables using SQL queries in DBMS

Exercise

Map the Schema for following ER Diagram and do the File Design.



Lesson 10 – Design Concepts along with Software Development

Software Design

Software Design sits at the technical kernel of software engineering. It is a problem solving process whose objectives to find and describe a way.

System Analysis determine **WHAT** has to be done and System Design determine **HOW** to do it.

Why Software Design is Important?

- Single word QUALITY.
- Design is the place where quality is fostered in software engineering.
- Design provides you with representations of software that can be accessed for quality.
- Without design, you risk building an unstable system – one that will fail when small changes are made, one that may be difficult to test.

Object Oriented Design Concepts

- **Encapsulation**: That idea of encapsulation is to hide how a class does its business, while allowing other classes to make requests of it.
- **Inheritance**: Allow Classes to inherit commonly used State and Behavior from other Classes.
- **Polymorphism**: It is an ability of an object to take on many forms.
- **Abstraction**: Abstraction is the process of refining away all the unneeded/unimportant attributes of an object and keep only the characteristics best suitable for your domain.
- **User Interface Classes:** Define all abstraction that are necessary for human computer interaction.
- **Business Domain Classes:** The classes identify the attributes and services that are required to implement some element of the business domain.

- **Process Classes:** Lower level business abstractions required to fully manage the business domain classes.
- **Persistent Classes:** Represent data stores that will persist beyond the execution of the software.
- **System Classes:** Implement software management and control functions that enable the system to operate and communicate within its computing environment and with the outside world.

Design Patterns

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software**, there are 23 design patterns which can be classified in three categories:

Creational, Structural and Behavioral patterns

1. Creational Design Patterns

These design patterns are all about class instantiation. These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.

- **Abstract Factory**
  Creates an instance of several families of classes
- **Builder**
  Separates object construction from its representation
- **Factory Method**
  Creates an instance of several derived classes

- **Object Pool**

  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use

- **Prototype**

  A fully initialized instance to be copied or cloned

- **Singleton**

  A class of which only a single instance can exist

2. Structural Design Patterns

These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

- **Adapter**

  Match interfaces of different classes.

- **Bridge**

  Separates an object's interface from its implementation.

- **Composite**

  A tree structure of simple and composite objects.

- **Decorator**

  Add responsibilities to objects dynamically.

- **Facade**

  A single class that represents an entire subsystem.

- **Flyweight**

  A fine-grained instance used for efficient sharing.

- **Private Class Data**

  Restricts accessor/mutator access

- **Proxy**

  An object representing another object

3. Behavioral Design Patterns

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Chain of responsibility**

  A way of passing a request between a chain of objects.

- **Command**

  Encapsulate a command request as an object.

- **Interpreter**

  A way to include language elements in a program.

- **Iterator**

  Sequentially access the elements of a collection.

- **Mediator**

  Defines simplified communication between classes.

- **Memento**

  Capture and restore an object's internal state.

- **Null Object**

  Designed to act as a default value of an object

- **Observer**

  A way of notifying change to a number of classes

- **State**

  Alter an object's behavior when its state changes

- **Strategy**

  Encapsulates an algorithm inside a class

- **Template method**

  Defer the exact steps of an algorithm to a subclass

- **Visitor**

  Defines a new operation to a class without change

Creational Design Patterns – Abstract Factory

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- The purpose of the Abstract Factory is to provide an interface for creating families of related objects, without specifying concrete classes.

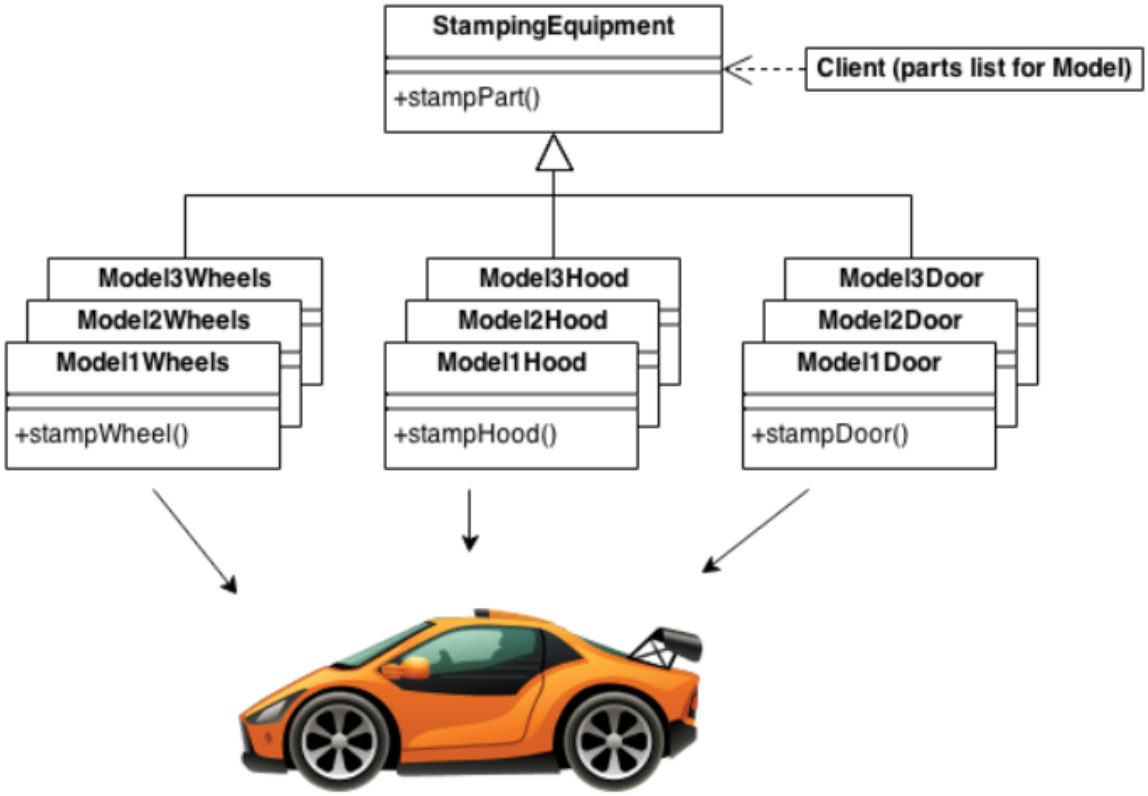


Figure 10.0.1 Abstract Factory

Creational Design Patterns – Object Pool

- Object pooling can offer a significant performance boost; it is most effective in situations where the cost of initializing a class instance is high, the rate of instantiation of a class is high, and the number of instantiations in use at any one time is low.
- The Object Pool lets others "check out" objects from its pool, when those objects are no longer needed by their processes, they are returned to the pool in order to be reused.
- By keeping reusable instances of objects in a resource pool, and doling them out as needed.

Creational Design Patterns – Singleton

- Ensure a class has only one instance, and provide a global point of access to it.
- Encapsulated "just-in-time initialization" or "initialization on first use".
- A single constructor, that is private and parameter less.

Behavioral Design Patterns – Chain of Responsibility

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
- Launch-and-leave requests with a single processing pipeline that contains many possible handlers.
- An object-oriented Linked List with recursive traversal. Use Pointers concept.

MVC Architecture

Model View Controller: Is a software architecture, which separates the logic from the user interface. This is achieved by separating the application into three parts Model, View and Controller. MVC is separation of concern. MVC is also a design pattern.

- **Model**: Represents the logical behavior of data in the application. It represents applications business logic. Interactions with Database. Model notifies view and controller whenever there is change in state.

- **View**: Provides the user interface of the application. A view is the one which transforms the state of the model into readable HTML.
- **Controller**: Accepts inputs from the user and instructs the view and model to perform the action accordingly.

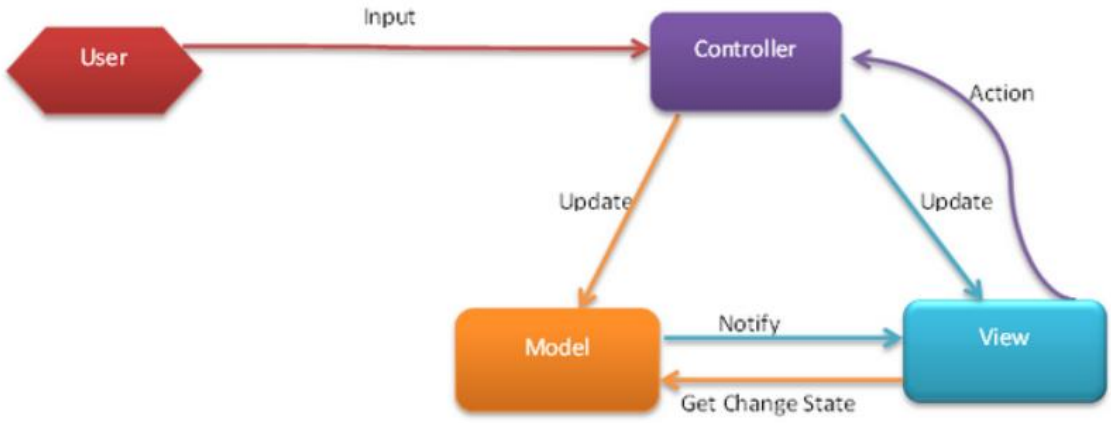


Figure 10.0.2 MVC Architecture

It should also be noted here, that your Controller acts as a bridge between your Model and the View. Because they, as them self, cannot perform any action.

The user must not be allowed to interact with Model himself, instead a Controller must be used to connect to the Model to get the data for the user's View that would be shown to him.

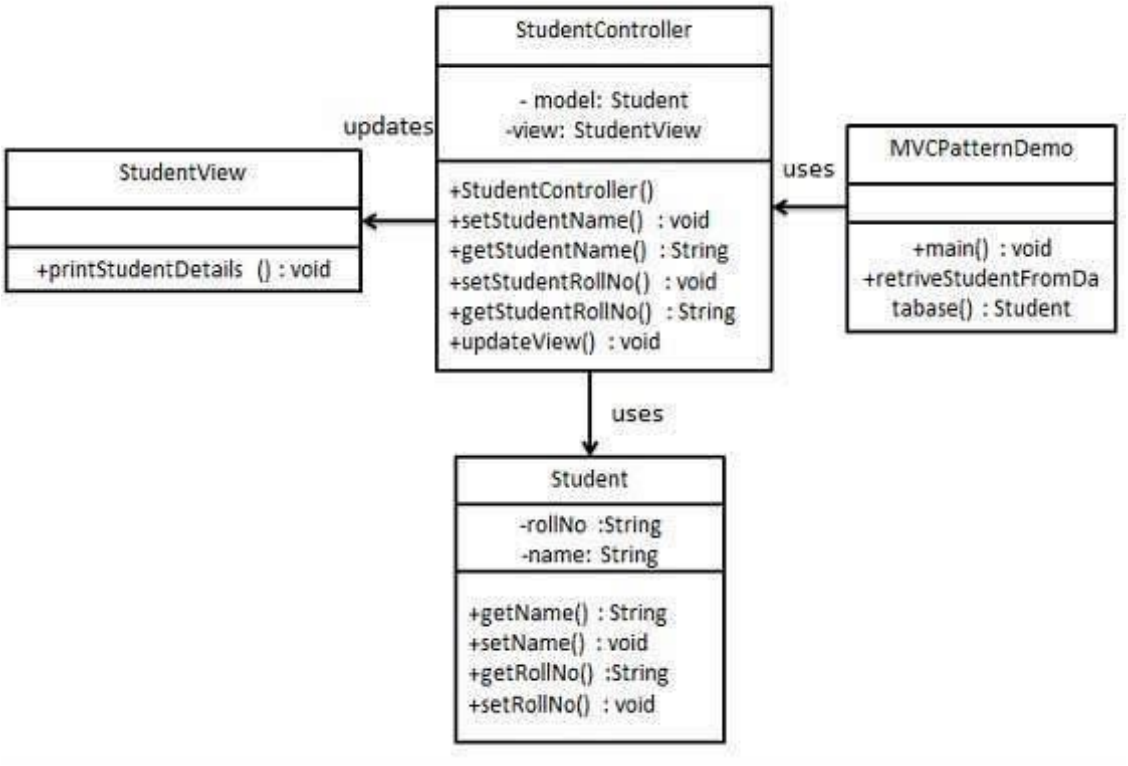


Figure 10.0.3 Model View Controller

MVC specially used in Web Development

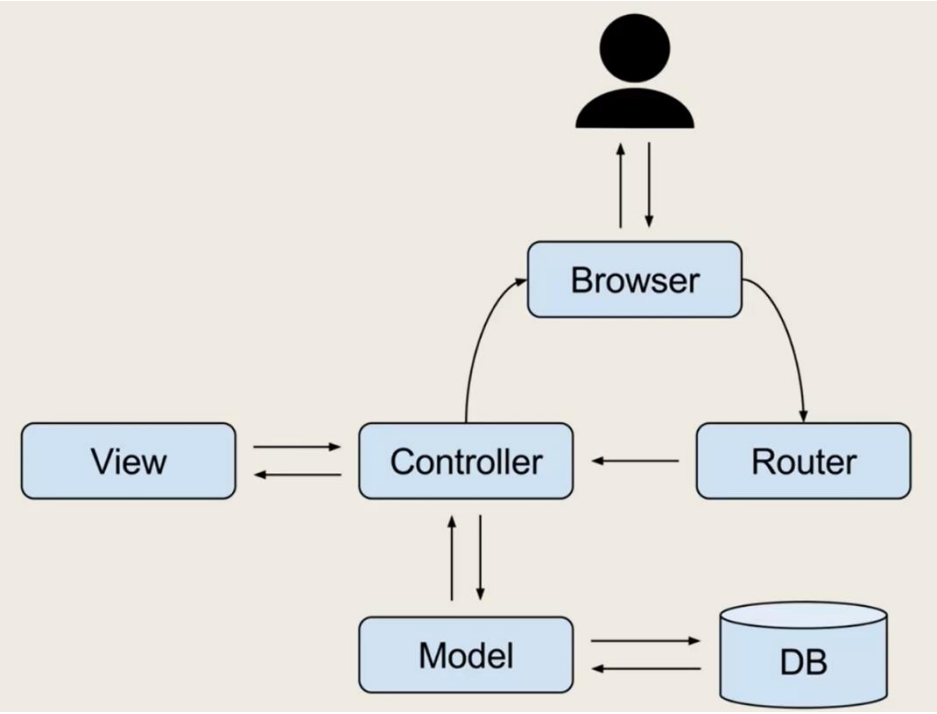


Figure 10.0.4 MVC in web development

User Interface Design

Creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

User Interface Design Principles (The Golden Rules)

1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent.

1) **Place the user in control:** User wanted a system that reacted to his/her needs and helped his/her get things done.

   - **Define interaction modes in a way that does not force a user into unnecessary or undesired actions**: Ex: Spell Check in Word Document.
   - **Provide for flexible interaction:** Different users have different interaction preferences. Ex: Keyboard commands, Mouse movements, a multi touch screen.
   - **Allow user interaction to be interruptible and undoable:** Ex: Undo any action.
   - **Streamline interaction as skill levels advance and allow the interaction to be customized:** Ex: It is worthwhile to design a macro mechanism.
   - **Hide Technical internals from the casual user:** The user should not be aware of the operating systems or other computer technology.

2) **Reduce the User's memory Load:** The more user has to remember, the more error-prone the interaction with the system will be.

   - **Reduce demand on short-term memory:** The interface should be designed to reduce the requirement to remember past actions, inputs and results.
   - **Establish meaningful defaults:** The initial set of defaults should make sense for the average user.
   - **Define shortcuts that are intuitive:** Ex: CTRL+ P to invoke the print function.

- **Disclose information in a progressive fashion:** Ex: After user pick the underlined function then all underlining options are presented.

3) **Make the Interface Consistent:** The interface should present and acquire information in a consistent fashion.

   - **Allow the user to put the current task into a meaningful context:** It is important is provide the indicators.
   - **Maintain consistency across a family of applications:** A set of applications should all implement the same design.
   - **If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so:** Ex: CTRL+S for save, the user expects that in every application he encounters.

To build an effective user interface "all design should begin with an understanding of the intended user, including profiles of their age, gender, physical abilities, education motivation, goals and personality".

Pay attention to what users do.

"Know the User, Know the Tasks"

Lesson 11 – Software Quality Assurance (SQA)

What is Software Quality Assurance (SQA)?

Software Quality Assurance (SQA) is simply a way to assure quality in the software. **Software Quality Assurance is a process which works parallel to development of a software.** It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. **Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.**

QA works out ways to **prevent possible bugs** in the process of software development. For instance, if a defect is found and fixed, there is no guaranteeing it won't pop back up. The role of QA is to identify the process that allowed the error to occur and re-engineer the system so that these **defects won't appear for the second time.**

The QA process verifies that the product will continue to function as the customer expects.

Software Quality Assurance Activities

1. Creating an SQA Management Plan:

Make a plan how to carry out the SQA throughout the project. Along with what SQA approach you are going to follow, what engineering activities will be carried out.

2. Set The Check Points:

SQA team should set checkpoints in different stages in the project to make sure the Quality of the Project being maintain. Evaluate the performance of the project on the basis of collected data on different check points. This ensures regular quality inspection and working as per the schedule.

3. Apply Software Engineering Techniques:

Software engineering techniques are the most important part to test your product. The software designer can prepare the project estimation using techniques like WBS (work breakdown structure).

4. Executing Formal Technical Reviews:

This can be carry out to evaluate the quality and design of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

5. Having a Multi- Testing Strategy:

Must not rely on a single testing approach Multiple testing strategies should be performed from all angles to ensure better quality.

6. Change Controlling

Use mix of manual methods and automated tools to have a strategy for change control.

7. Measuring Change Impact:

In this phase, if any bug or error is found then the QA team will report it to the Software Engineers so that they can fix it.

8. SQA Audits

SQA audit checks that error reported by the QA team were actually performed or not by the Software Engineers.

9. Records and Reports Maintenance

The audit results, test results, change requests documentation, review reports should be kept for future reference.

10. Manage Good Relations:

Managing good relation between developer and tester is also a very important factor because they both think that they are superior to each other.

Software Quality Assurance Techniques

Auditing is the most important technique that is widely used in the Industry. Other than Auditing follow techniques can be also use.

- **Auditing:** Auditing involves inspection of the work products and its related information.
- **Reviewing**: Software is examined by both the internal and external stakeholders.
- **Code Inspection:** It is the most formal kind of review that does static testing to find bugs and avoid defect growth in the later stages. It is done by a trained mediator/peer. Not the same developer who develop the Code.
- **Design Inspection:** Focus on Functional and Interface specifications, Requirement traceability, Logic, Performance, Error handling and recovery.
- **Functional Testing:** Focuses on testing the system specifications or features.
- **Walkthroughs:** In Software walkthrough or code walkthrough development team to go through the product and raise queries, suggest alternatives or any other issues.
- **Path Testing:** It is a whit box technique where the complete branch coverage is ensured by executing each independent path at least once.
- **Stress Testing:** This type of testing is done to check whether the system can function under heavy load.
- **Six Sigma:** Six Sigma is a quality assurance approach that aims at nearly perfect products or services. (The software is 99.76 % defect free)

Benefits of Software Quality Assurance

- SQA produce high quality software.
- High quality application saves time and cost.
- No need maintenance for long time.
- High quality commercial software increase the market demand.
- Improving the process of creating software.

Software Testing

It is the process of executing a system in order to find bugs (errors) in the Software to get that fixed. Testing is helps to runs the software the way it is expected and designed for.

Software testing is a process of verifying and validating that the software is,

- Error Free
- Meets the technical requirements
- Meets the user requirements effectively and efficiently
- Handling all the exceptional and boundary cases.

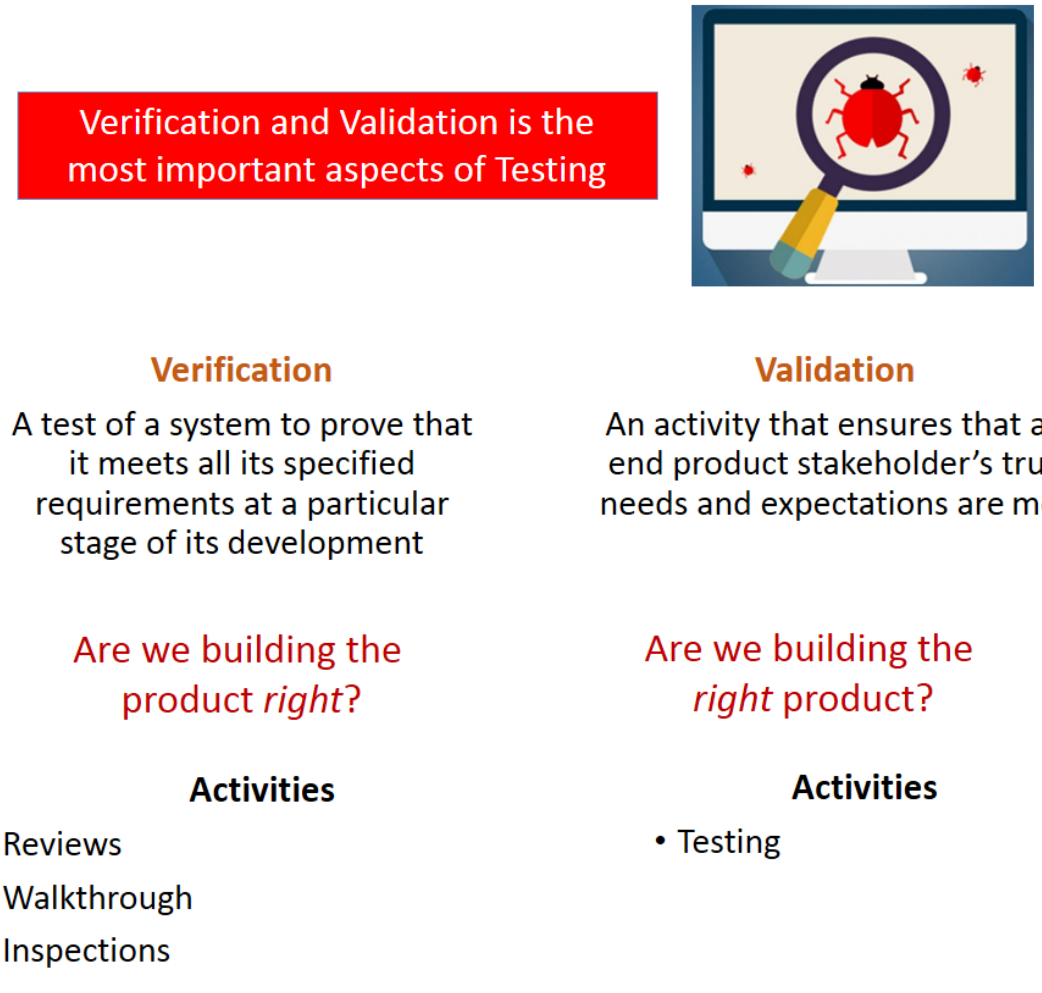
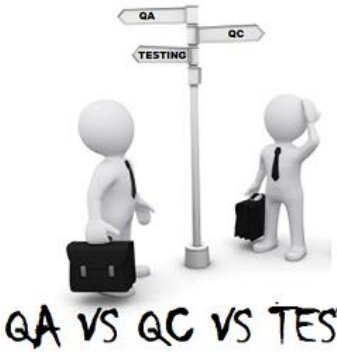


Figure 11.0.1 Difference between Verification and Validation

Difference between Quality Assurance and Testing



Quality Assurance applied throughout the software process to prevent possible bugs in the process of software development.

Quality Assurance is Process Oriented and Proactive

Quality Control detects bugs by inspecting and testing the product. Validating that the product meets those requirements.

Quality Control is Product Oriented and Reactive

Testing is a subset of Quality Control. It is the process of executing a system in order to find bugs (errors) in the Software to get that fixed.

Figure 11.0.2 QA, QC and Testing

Software Quality Assurance	Software Testing
Ensure that Software is being developing according to the Specification	The Process of executing the system in order to find errors
Assure the quality and meeting the requirements	Software inspection and bug finding
Process Oriented	Product Oriented
Preventive	**Corrective**

Table 11.0.1 Software Quality Assurance vs Testing

Software Testing Life Cycle

Software Testing Life Cycle is a process which has specific steps to ensure that the quality of the Software have been met. Different Organizations may have different phases and following are the common steps to be follow.

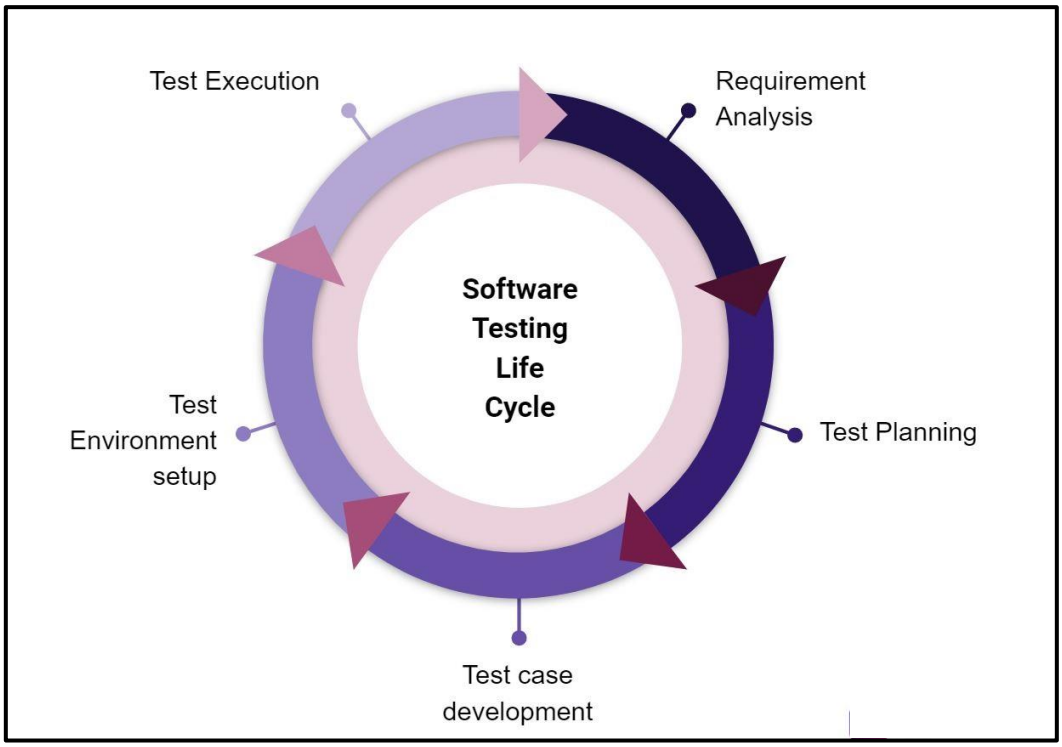


Table 11.0.2 Software Testing Life Cycle

1. Requirement Phase:

During this phase, QA team studies the requirements from a testing point of view to identify the testable requirements.

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.

2. Test Planning Phase:

Identify the activities and resources which would help to meet the testing objectives.

- Preparation of test plan/strategy document for various types of testing.
- Test tool selection.
- Test effort estimation.
- Resource planning and determining roles and responsibilities.
- Training requirement.

3. Test Case Development Phase:

This phase involves the creation, verification and rework of test cases & test scripts.

- Create test cases, automation scripts
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

4. Test Environment Setup Phase:

Based on the Customer's environment Test environment setup for the software and hardware conditions of the product.

- Prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data.
- Perform Smoke Test (verified to ensure the stability of the application for further testing) on the build.

5. Test Execution Phase:

Testing carried out based on the test plans and the test cases. Bugs will be reported back to the development team for correction and retesting will be performed.

- Execute tests as per plan.
- Document all the test results, and log defects for failed cases.
- Map defects to test cases in Requirement Traceability Matrix (RTM).
- Retest once the development team fixed the defects.

6. Test Cycle Closure Phase:

QA team analyze testing artifacts and performance of the testing to identify new strategies for the future testing. This is a learning process.

- Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality.
- Document the learning outcomes from the software.

How to write a Test Case?

Test cases help guide the QA team through a sequence of steps to validate whether a software application is free of errors and working as required by the end user. The test case includes specific variables or conditions, so the QA team can compare expected and actual results to determine whether a software is functioning as the end user's requirements.

Test Cases for Login Page

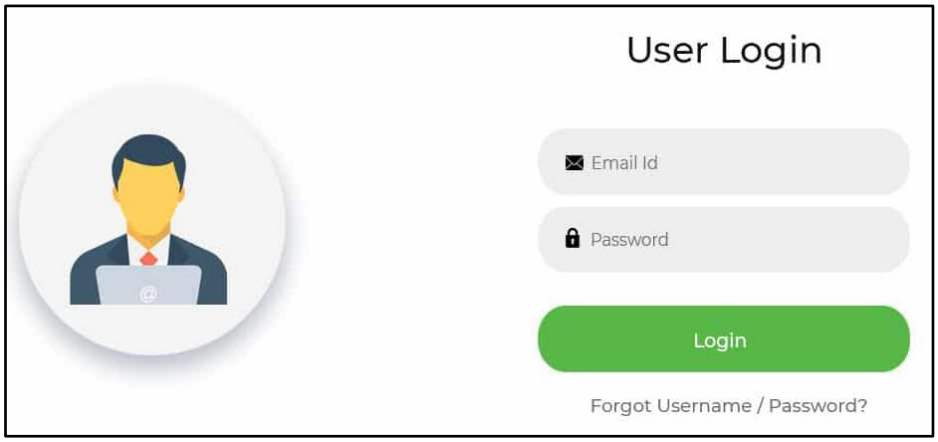


Figure 11.0.3 Test Cases for Login Page

The format of Standard Test Cases

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail

Table 11.0.3 Format of Standard Test Cases

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T01	Verify that cursor is focused on “Username” text box on the page load	After Loading the Login Page	-	Cursor if focused on username field	As Expected	Pass
T02	Verify that the Login screen contains elements such as Username, Password, Login button, Forgot Username/ Password link	After Loading the Login Page	-	Verify the components in Login screen	As Expected	Pass
T03	Verify that Tab functionality is working	Press the tab button	-	Cursor moves to	As Expected	Pass

	properly or not			Password field		
T04	Verify that the password is in masked form when entered	Enter some dummy values	1234	Password field is masked by *	As Expected	Pass
T05	Verify if the password can be copy-pasted or not.	Copy paste a value to the password field	1234	Cannot paste the value to the password field	Not as Expected	Fail
T06	Verify that User is able to Login with Valid Credentials	Enter Email and Password . Then click Login button	Email= keshan@gmail.com Password= lec789	User should Login into the system	As Expected	Pass
T07	Verify that User is not able to Login with invalid Username and invalid Password	Enter Email and Password . Then click Login button	Email= kehan@gmail.co Password= lec1234	Display Error Message Your Email or Password is Wrong	As Expected	Pass
T08	Verify that User is not able to Login with Valid Username	Enter Email and Password . Then click	Email = keshan@gmail.com Password = 45789	Display Error Message Your Email or Password is Wrong	As Expected	Pass

	and invalid Password	Login button				
T09	Verify that User is not able to Login with invalid Username and Valid Password	Enter Email and Password . Then click Login button	Email = keshan@gmail.com Password = lec789	Display Error Message Your Email or Password is Wrong	As Expected	Pass
T10	Verify that User is not able to Login with blank Username or Password	Enter Email and Password . Then click Login button	Email = Password = 45789	Display Error Message Please enter your	As Expected	Pass
T11	Verify that User is not able to Login with inactive credentials	Enter Email and Password . Then click Login button	Email = keshan@gmail.com Password = lec789	Display Error Message Your account is Inactive	As Expected	Pass
T12	Verify that there is a only 3 unsuccessful login attempts	Enter Email and Password . Then click Login button	Email = kes@gmail.com Password = 45789	Display Error Message Your Login attempts are over. Your	As Expected	Pass

				account Lock		
T13	Verify that User should be able to login with the new password after changing the password	Enter Email and Password . Then click Login button	Email = keshan@gmail.com Password = lec123	User should Login into the system	As Expected	Pass
T14	Verify that User should not be able to login with the old password after changing the password	Enter Email and Password . Then click Login button	Email = kes@gmail.com Password = lec789	Display Error Message Your Email or Password is Wrong	As Expected	Pass
T15	Verify that User is redirected to appropriate page after successful login	Enter Email and Password . Then click Login button	Email = keshan@gmail.com Password = lec123	User should able to see the Home Menu	As Expected	Pass

Table 11.4 Test Cases for Login Page

Software Testing Techniques (Methods)

Black box testing use to test the software without knowing the internal structure of code or program

White box testing use to test the software with the knowledge of internal structure of code or program

Black Box Testing

- Also known as Functional Testing. Programming knowledge or Implementation knowledge is not required.
- The3 main aim of this testing to check on what functionality is performing by the system under test.
- The black box testing is used to find,
    - Interface errors
    - Incorrect Functions that lead to undesired output
    - Missing Functions
    - Incorrect Outputs

An Example for Black Box Testing

- Testing User Name and Password in a simple Login screen.
- This test will check the input and output.

- A user can log to the system when inputs the correct username and correct password.
- A user receives an error message when enters incorrect username and incorrect password.

Figure 11.0.4 Black Box Testing - Example

White Box Testing

- Also known as Structural Testing or Glass Box Testing. Knowledge of internal working structure (Coding of software) is required.
- In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.
- The white box testing is used to find
  - Internal Security Holes
  - Testing of each statement
  - Functionality of conditional loops

Black Box Testing Technique - Equivalence Partitioning

This allows QA team to separate test conditions into partitions (classes). Test cases should be designed according to the different input domains created.

Ex: Suppose you want to test a input box which is accepting numbers from 1 to 100. There is no use in writing 100 test cases for all 100 valid input numbers plus other test cases for invalid data.

Using the Equivalence Partitioning method above test cases can be divided into three sets of input data called classes.

Testing Solution:

1. One input data class with all valid inputs. You can take a single value from range 1 to 100 as a valid test case.

2. Input data class with all values below the lower limit. Any value below 1, as an invalid input data test case.

3. Input data with any value greater than 100 to represent the third invalid input class.

Black Box Testing Technique - Boundary Value Analysis

Boundary testing is the process of testing between extreme ends or boundaries between partitions (classes) of the input values. More software **errors occur at the boundaries** of the input partitions. 'Boundary Value Analysis' Testing technique is used to identify errors at boundaries.

Ex: Suppose you want to test an input box which is accepting numbers from 1 to 100.

In this case you can test boundary values using boundary value analysis.

Testing Solution:

1. Test exactly the input boundaries of input domain. Test value 1 and 100.

2. Test data with values just below the extreme edges of input domains. Test 0 and 99.

3. Test data with values just above the extreme edges of the input domain. Test 2 and 101.

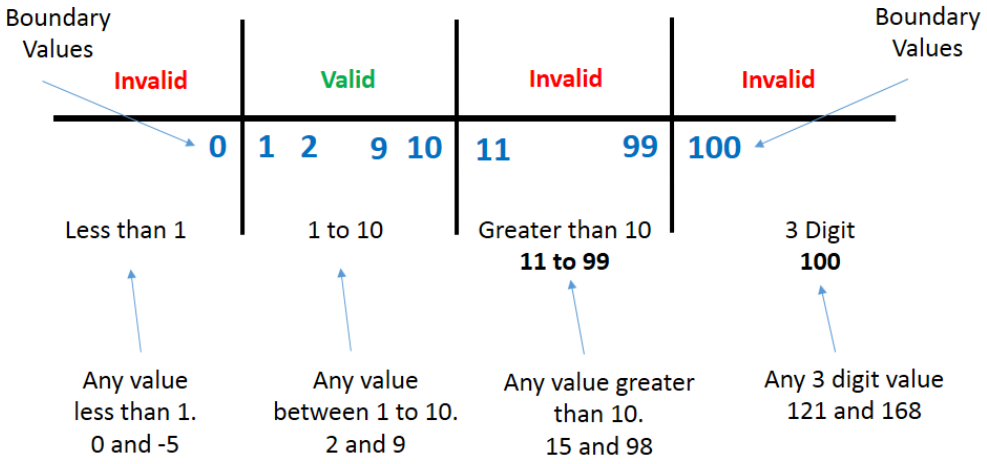
Example for Equivalence Partitioning & Boundary Value Analysis

Consider about a text box which accepts number of pizza. Maximum 10 pizza can be order and 1 to 10 is considered as valid inputs.

Here are the test conditions

- Any Number greater than 10 is invalid.
- Any Number less than 1 is invalid.
- Numbers 1 to 10 are considered valid
- Any 3 Digit Number say 100 is invalid.

We cannot test all the possible numbers. Then the number of test cases will be more than 100. Ex: You have to test 1 to 10. Then values greater than 10, less than 1, more than 2-digit number. Therefore, we can use Equivalence Partitioning technique.



Software Testing Levels (Types)

Functional Testing Types

- Unit Testing
- Integration Testing
- System Testing
- Sanity Testing
- Smoke Testing
- Interface Testing
- Regression Testing
- Beta/Acceptance Testing

Non Functional Testing Types

- Performance Testing
- Load Testing
- Stress Testing
- Volume Testing
- Security Testing
- Compatibility Testing
- Install Testing
- Recovery Testing
- Reliability Testing
- Usability Testing
- Compliance Testing
- Localization Testing

Functional Testing Types

Unit Testing

Test individual units/components of the software. The purpose is to validate that each unit or component of the software performs as designed. It is typically done by the Software Engineer and not by QA Engineer.

Integration Testing

Individual units are combined and tested as a group. The aim of this level is to expose errors in the interaction between integrated units.

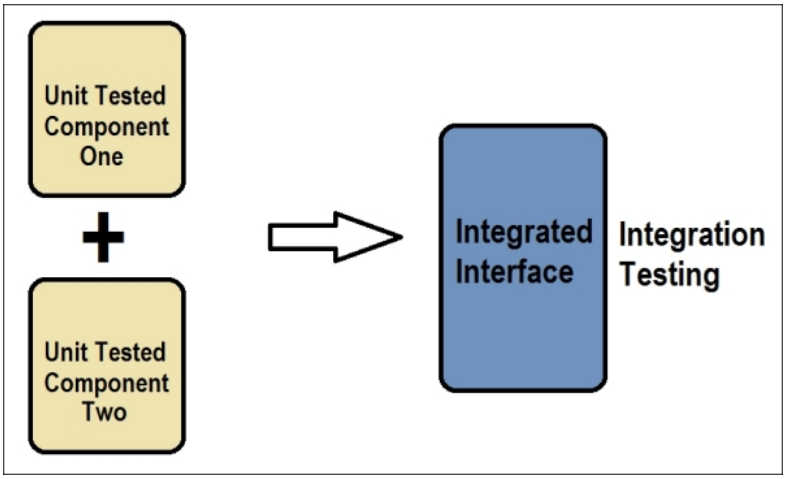


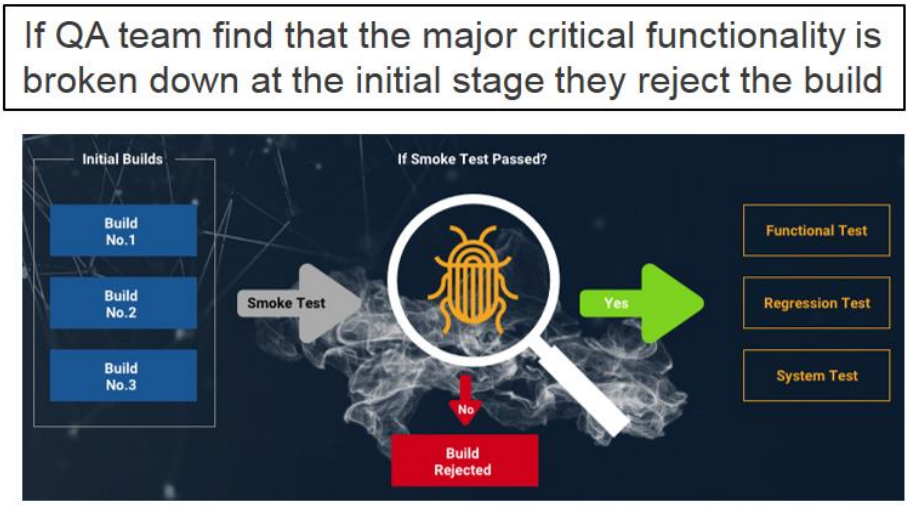
Figure 11.0.5 Integration Testing

System Testing

The entire software is tested as per the requirements specification. It is a Black-box type testing that is based on overall requirement specifications and covers all the combined parts of a system.

Smoke Testing

Whenever a new build is provided by the development team then the QA team validates the build to ensures that no major issue exists. This testing is ensuring that the build is stable and a detailed level of testing can be carry out further.



Acceptance (Alpha) Testing

Alpha testing is a type of acceptance testing which is perform to identify all possible bugs before releasing the software to real environment. This test is performing in the development environment by the developers and QA team (in the development site).

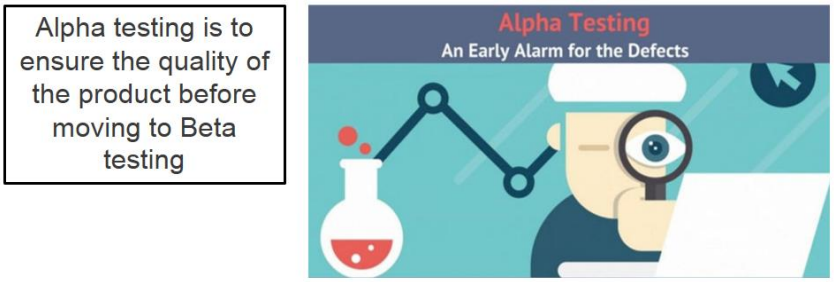


Figure 11.0.6 Acceptance (Alpha) Testing

Acceptance (Beta) Testing

An Acceptance Test is performed by the customer and users to verifies whether the system build as per the business requirements. This test is performing in the real environment by the real users (in the Client site).

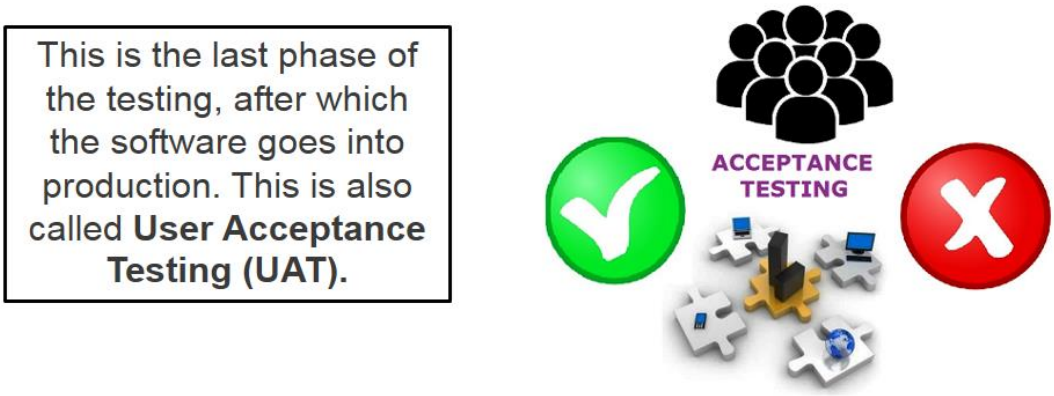


Figure 11.0.7 Acceptance (Beta) Testing

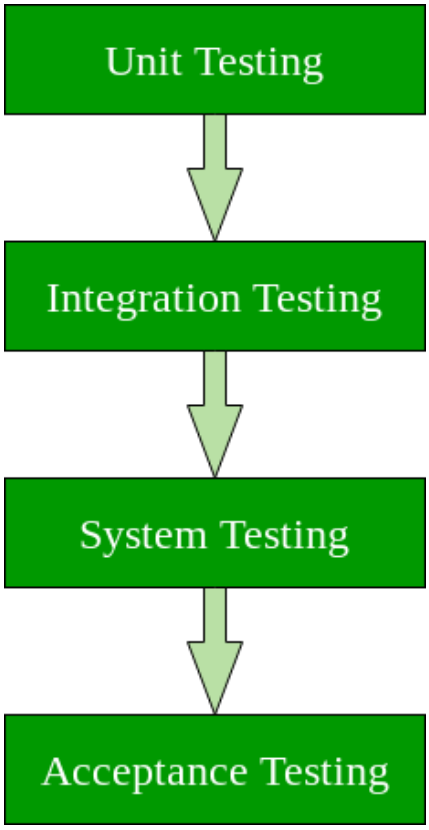


Figure 11.0.8 Software Testing Levels

Non Functional Testing Types

Performance Testing

- Performance Testing is done to check whether the system meets the performance requirements. Different performance and load tools are used to do this testing.

Load Testing

- To check how much load or maximum workload the software can handle without any performance deviations.

Stress Testing

- Software is stressed beyond its specifications in order to check how and when it fails.
- This is performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to the system or database load.

Security Testing

- To check how the software is secure from internal and external threats.
- This testing includes how much software is secure from the malicious program, viruses and how secure and strong the authorization and authentication processes are.

Compatibility Testing

- This testing validates how software behaves and runs in a different environment, web servers, hardware, and network environment.

Exhausting Testing

Exhaustive testing is a test approach in which all possible data combinations are used testing.

In this method all the types of inputs are given to the software to check the outputs.

Exhaustive testing is the process of testing for absolutely everything just to make sure that the software won't crash in any situation.

Ex: Consider the password field which accepts 3 characters only.

- If you take the alphabet only. You need to check 26 * 26 * 26 inputs
- If you take numbers and special characters also. It is 256 * 256 * 256 inputs

Exhausting Testing is not possible for complex large project. You cannot test all the possible inputs and combinations

Top Testing Tools

- **Bugzilla:**

  It is a Bug-tracking system which allow QA team to keep track of outstanding bugs, problems, issues, enhancement and other change requests in their products effectively.

- **Selenium:**

  Selenium is a testing framework to perform web application testing across various browsers and platforms like Windows, Mac, and Linux.

- **TestRail:**

  Efficiently manage manual and automated test cases, plans, and runs. Get real-time insights into testing progress with informative dashboards, metrics, and activity reports.

- **Loadrunner:**

  It is a load testing tool for Windows and Linux, which allows testing the web application efficiently. It helpful testing tool to determining the performance and result of the web application under heavy load.

Lesson 12 – Software Maintenance

What is Software Maintenance?

This is the process of modifying the software or system or component of the system after deliver to the customer. The main aims of Software Maintenance are correct faults, improve performance or other attitudes, or adapt to a change in the environment. **Software Maintenance is concerned with modifying software once it is delivered to a customer.**

What is meant by Software Maintenance for Organization?

- A major economic importance.
- A substantial applications backlog.
- Starting for late 1960s, Software Maintenance started to be recognized as a significant activity.

According to statistics,

"when it comes to software,

more than 60% of costing involves maintenance".

Why Software Maintenance?

Because Software Change all the time and Software Change is Inevitable

- New customer requirements emerge when the software is used.

  Ex: Customer may need a new function like Face Recognition for Login or QR code for payments.

- The Business environment changes.

  Ex: Because of the Covid19 all the Banking and Financial Systems have to change.

- Errors must be updated.

  Ex: When users use the Software over the time they can identify errors in the Software.

- The Performance and Reliability must be improved.

  Ex: The software security holes must update for the latest encryption modes.

- Changes of Market Conditions

  Ex: Policies, Taxation and newly introduced constraints of Government.

- Host Modifications

  Ex: If Customer want to change the hardware and/or platform (such as operating system).

- Organization Changes

  Ex: If there is any business level change at client end, such as reduction of organization strength, acquiring another company.

Software Maintenance Types

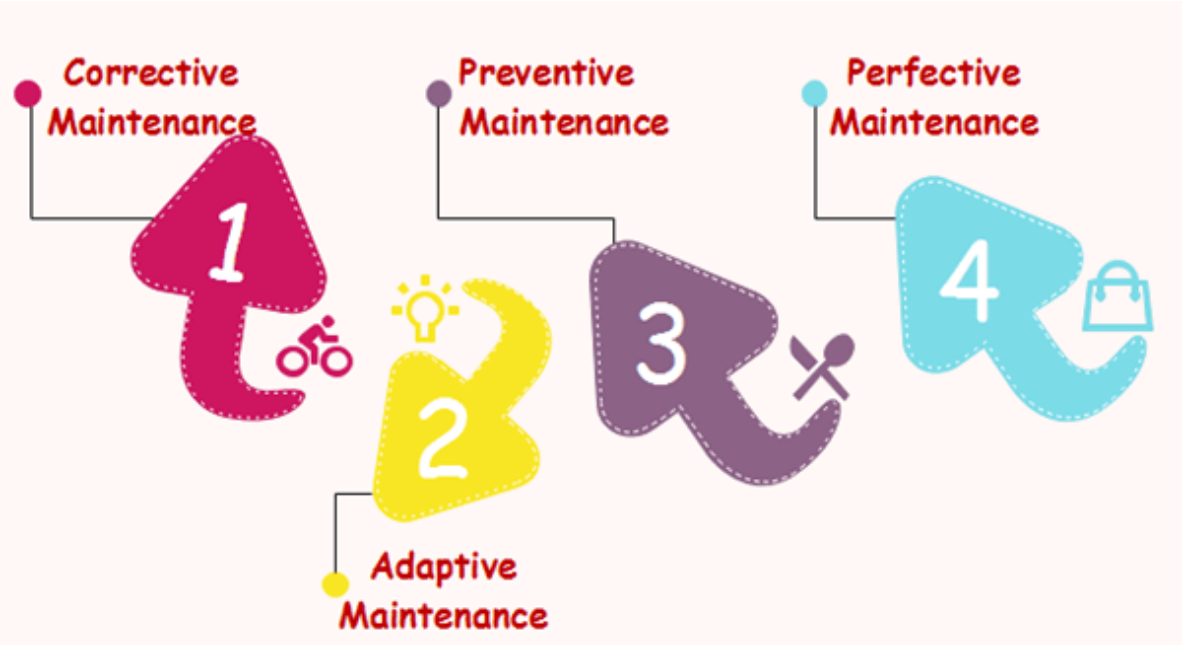


Figure 12.0.1 Software Maintenance Types

- **Corrective Maintenance:**

The identification and removal of faults in the software.

> *Ex: After few months' user found that TP text field accepts special characters. You have to immediately fix that bug.*

- **Adaptive Maintenance:**

Changes needed as a consequence of operating system, hardware or DBMS changes. To keep the software product up-to date and tuned to the ever changing world of technology and business environment.

> *Ex: After one year new SQL Server version introduced to the market. Therefore, we have to change the current database to adapt with new version.*
>
> *Ex: After few months' customer decided to use MAC OS instead of Microsoft. Therefore, we have to change the current software to match with the new operating system.*

- **Preventative Maintenance:**

Changes made to software to make it more maintainable. Modifications and updates carry out to prevent future problems of the software. It aims to attend problems, which are not significant at this moment **but may cause serious issues in future.**

> *Ex: Customer may face cyber-attacks in future, therefore you take actions now to prevent those future attacks.*

- **Perfective Maintenance:**

Changes required as a result of user requests. It includes new features, new user requirements for refining the software and improve its reliability and performance.

> *Ex: Customer may request a token id when user's login to the system to make sure the security.*

Software Cost Components

- Hardware and Software costs.
- Travel and training costs.
- Effort costs
  - The salaries of engineers involved in the project
  - Social and insurance costs

- Effort costs must take overheads into account
  - Cost of building, heating, lighting
  - Costs of networking and communications
  - Costs of shared facilities

Software Maintenance Cost

The cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

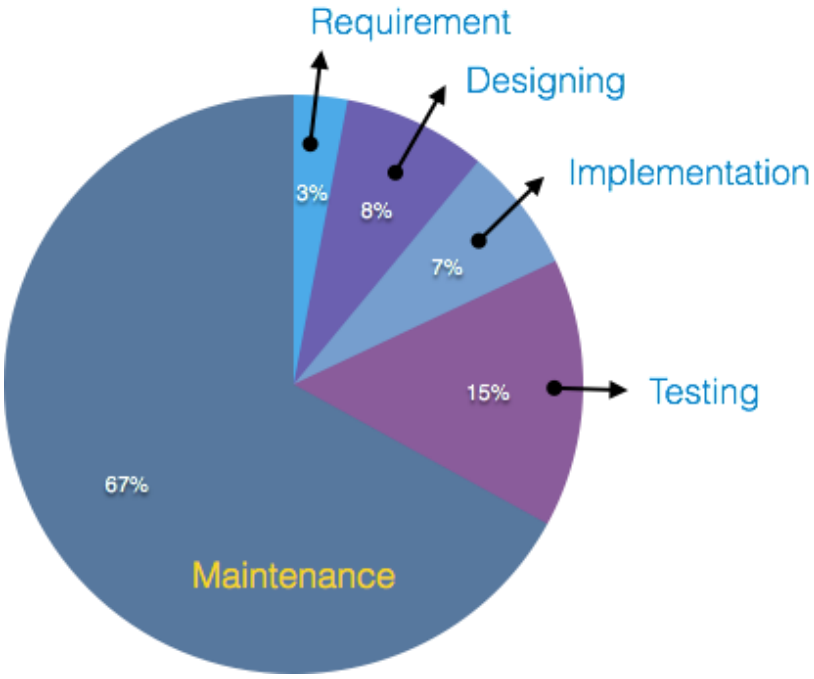


Figure 12.0.2 Software Maintenance Cost

Why Software Maintenance Cost is High?

- The standard age of any software is considered up to 10 to 15 years.
- Older software, cannot keep themselves challenging against newly coming enhanced software on modern hardware.
- As technology advances, it becomes costly to maintain old software.

- Most maintenance engineers are new and use trial and error method to rectify problem. Their training cost is high.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software Maintenance Activities

IEEE provides a framework for sequential maintenance process activities.

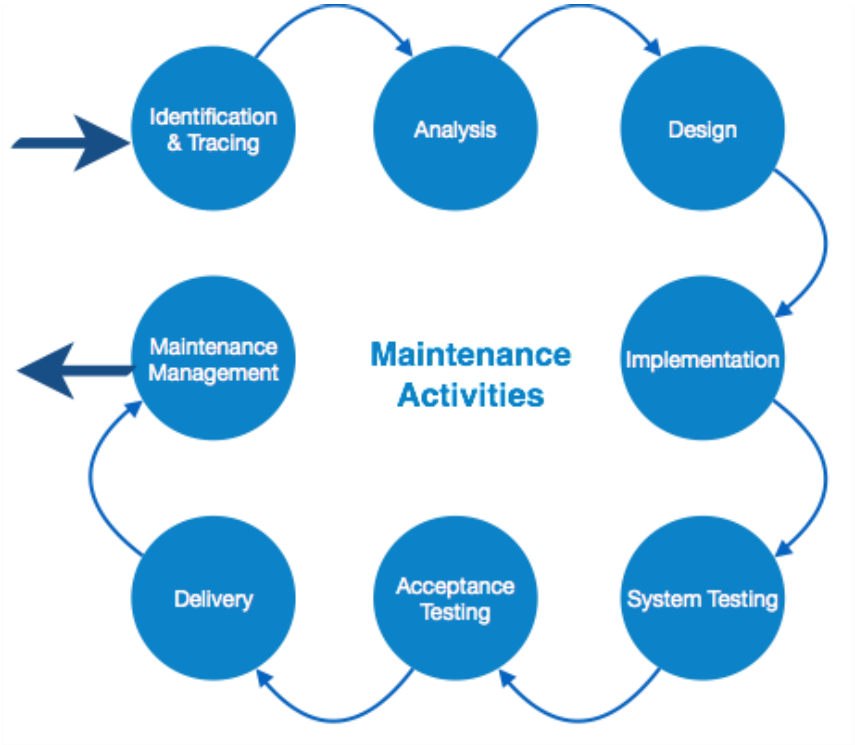


Figure 12.0.3 Software Maintenance Activities

- **Identification & Tracing** - Identify the requirement to be modify or maintenance. It may be generated by the user or software.
- **Analysis** - The modification is analyzed for its impact on the software in terms of safety and security. All the modification record in the SRS and the cost of modification/maintenance is analyzed and estimate.
- **Design** - If it is needed to replaced or modified a function or a requirement, then new modules should be design. Test cases has to create for validation and verification.

- **Implementation** - The new modules are coded with the help of structured design and Developer must do the Unit Testing in parallel.
- **System Testing** - Integration testing should carry out with newly created modules and also between new modules and the system. Finally, the system testing performed for the entire system.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of end users. If the end user's complaints, those complaints address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system.
- **Maintenance Management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

Software Re-Engineering

Reengineering is a rebuilding activity. When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Software Re-Engineering is concerned with re-implementing legacy system to make them more maintainable. It may be the only viable way to ensure that legacy system can continue.

Legacy System is typically:

- Very Old and Large and has been heavily modified
- Based on the old technology
- Documentation not be available
- None of original members of the software development team may still be around

- The software is often at the core of the business and replacing it would be a Huge expense.

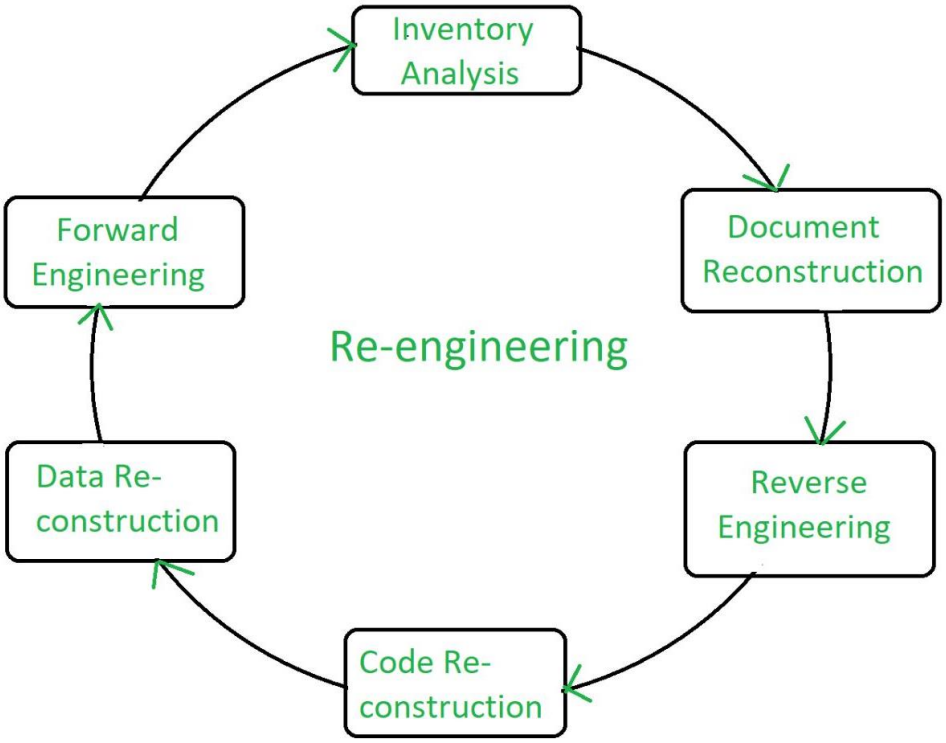


Figure 12.0.4 Software Re-Engineering

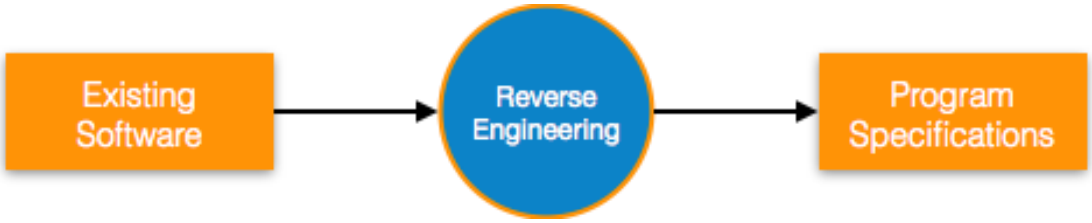
Re-Engineering Process

- Decide what to Re-Engineer. Is it whole software or a part of it?
- Perform Reverse Engineering, in order to obtain specifications of existing software.
- Restructure Program if required. For example, changing function-oriented programs into object-oriented programs.
- Re-structure data as required.
- Apply Forward engineering concepts in order to get re-engineered software.

Reverse Engineering

- Suppose in the existing Software the design documentation is not available. In that case Designers do the reverse engineering by looking at the code and try to figure out the software design.

- If the design can be figure out, then they try to conclude the specifications.

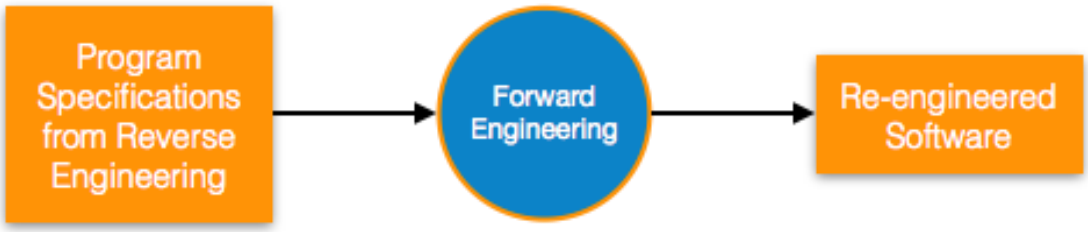


Program Restructure

- It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Also data can be restructured if it is needed. As an example normalizing the tables.

Forward Engineering

- Once the specification is concluded from Reverse Engineering, Forward engineering can carry out to develop the software. It assumes that there was some software engineering already done in the past.
- Forward engineering is same as software engineering process with one difference – it is carried out always after reverse engineering.



Business Process Reengineering (BPR)

Business Process Reengineering involves the radical redesign of core business processes to achieve dramatic improvements in productivity, cycle times and quality. BPR starts with the important assets of the organization: Core competency, Personnel, Skills, Knowledge, Customer Relationships, Market Image and Information.

It attempts to recombine these assets in new and innovative ways to help companies gain Competitive Advantage.

Core Concepts

- Fundamental Analysis
  - Identify what must be done
  - Identify performance measures
- Radical Redesign
  - Be creative and practical – It must work
  - Develop options -Solutions
- Dramatic Improvement
  - Set aggressive achievable target
- Eliminate non-value adding functions through process redesign.

How do we add values?

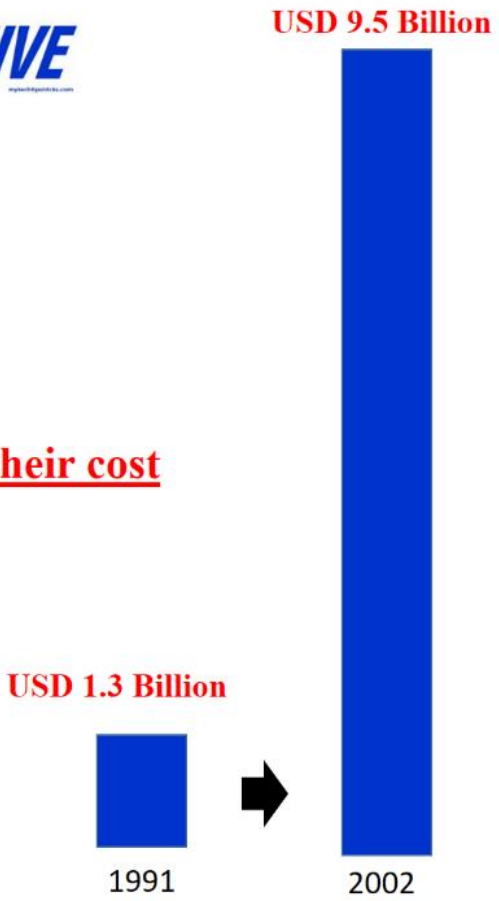
- Eliminate
- Simplify
- Integrate
- Automate

Real World Examples for BPR - 1

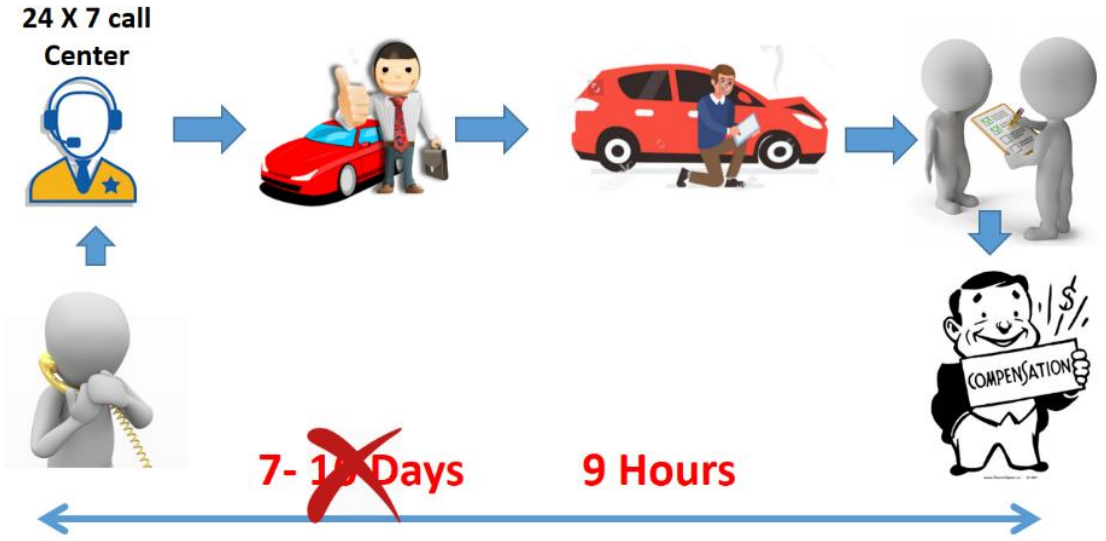
PROGRESSIVE

- An automobile Insurer
- Operates in United States
- Country's third largest insurer

Reinvented claim process to **lower their cost** and **boost customer satisfaction**



Introduced "Immediate response Claims handling"

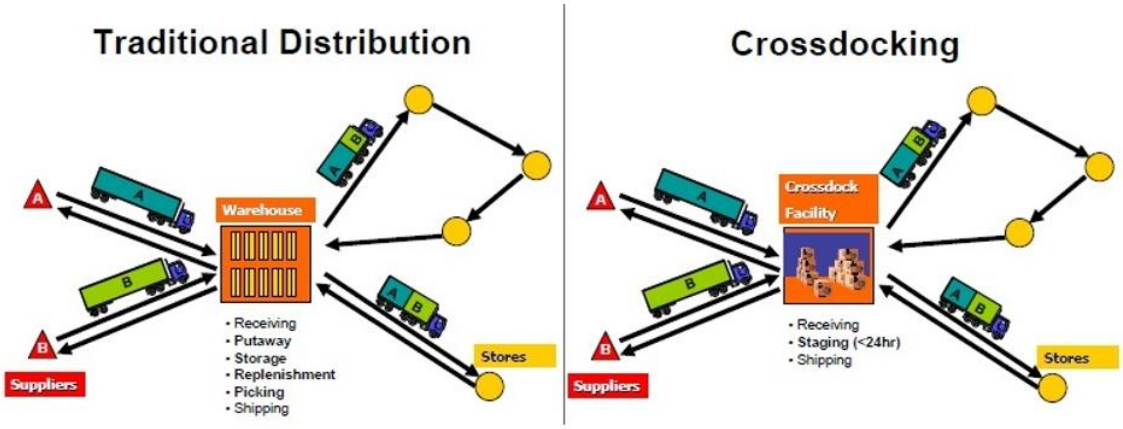


Real World Examples for BPR - 2

Walmart

$ 44 million (1972) → $ 44 billion (1992)

Higher profits, faster growth, lower prices



Real World Examples for BPR - 3

Ford

Traditional Account Payable Process

- 500 Employees
- High Cost
- Match 14 items before payments
- Compare to Mazda 5 times bigger than Ford but only 5 employees involved

→

After BPR the Account Payable Process

- Employees cut down by 75%
- No Invoice
- Match only 3 items

Some more Real World Examples for BPR

Introduced "Business Process Executive" (BPE) to increase the on time the product to market. 75% reduction in time to market

- Increased on-time deliveries
- Increased customer satisfaction
- $9 billion savings

Introduced Build to Order Process. This means for production and supply chain, that there is no action until a customer order comes

Toyota Just In Time.

In this production process, the company uses its supply chain in such a manner that only the parts that are needed to manufacture vehicles are received on time.

Advantages of BPR knowledge to Software Engineers

- Can design new process
- Changing the business environment
- Eliminate
- Increase the efficiency
- Make better, creative software for the customer and it may also be practical for modern business environment.

Lesson 13 – Software Configuration Management

What is Software Configuration Management?

Software changes frequently. We need to manage evolving systems as they evolve, many different versions of the software are created. Software Configuration Management is an umbrella activity that is applied throughout the software process.

It is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal is to increase productivity with minimal mistakes.

Why do we need Configuration Management?

- Since multiple developers and QA engineers are working on software, it is continually updating.
- Multiple version and authors are involved in a software, and the team is geographically distributed and works concurrently.
- Changes in user requirement, policy, budget, schedule need to be recorded and maintain.
- Helps to develop coordination among stakeholders.
- Need a control over the costs involved in making changes to a system.

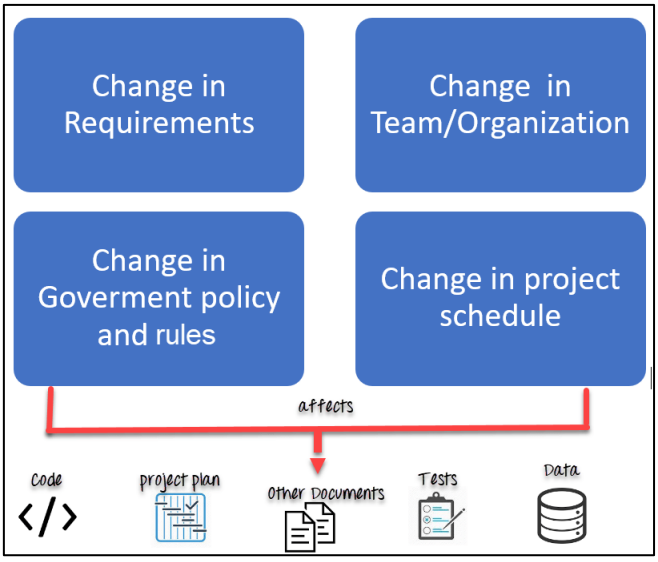


Figure 13.0.1 Configuration Management

Configuration Management Repository

- Today Configuration Items, details maintain in a project database or repository.
- The CM repository is the set of mechanisms and data structures that allow team to manage changes in effective way.
- CM Repository provides a hub for the integration of software tools and is central to the flow of the software process.

Configuration Management Repository Features

- **Versioning:** Many versions of individual work products will be created and the repository must be able to save all of these versions.
- **Dependency Tracking and Change Management:** The repository manages a wide variety of relationships among the data elements stored in it.
- **Requirements Tracing:** Ability to track all the design and construction components and deliverables.
- **Audit Trails:** Establishes additional information about when, why and by whom changes are made.

Version Control

A component of software configuration management. Version control, also known as revision control or source control. It is the management of changes to documents, computer programs, large web sites, and other collections of information.

Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged. In addition, Version Control can act as an issues tracking (bugs tracking) system that record and track the status of all outstanding issues.

Version Derivation Structure

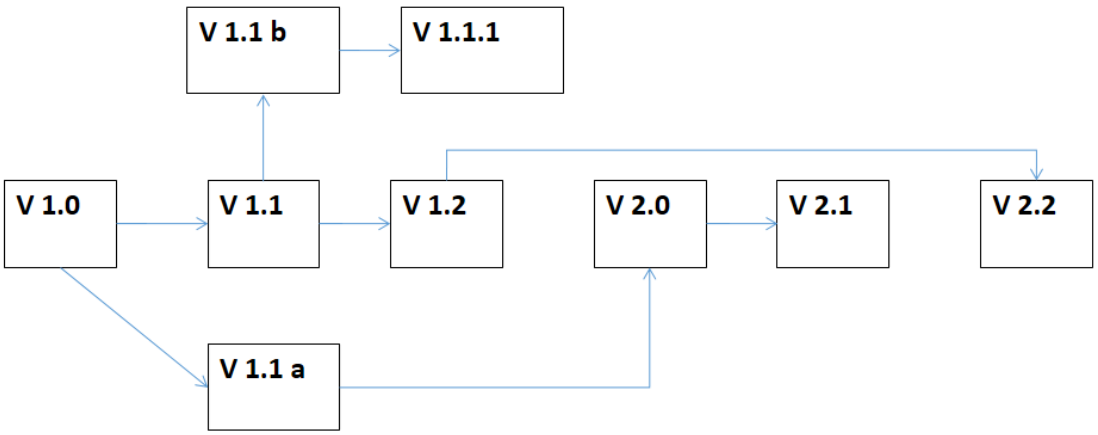


Figure 12.2 Version Derivation Structure

System Release Management

- Release Management is the process of managing, planning, scheduling and controlling a software build through different stages and environments; including testing and deploying software releases.
- It is not just a set of executable programs.
- May also include:
  - Configuration Files defining how the release is configured for a particular installation.
  - Data files needed for system operations.
  - Packages and other information.

System Release Problems

- Customer may not want to use the new release.
- They may happy with their existing system as the new versions may provide unwanted functionality.
- All files required for a release should be recreated when a new release is installed.

Methods used to convert to a new system

- Parallel Running
- Direct Approach
- Pilot Approach
- Phase in Approach

Parallel Running

This involves the operating of the old system and the new system simultaneously for a period of time. During this time each output of the new system is compared with the relevant output of the old system.

- **Advantages**
    - Provides high degree of protection
- **Disadvantages**
    - Cost of duplicating each computer related activity.
        - Consume more time and resources

Direct Approach

In this method conversion takes place all at once during short period of time and therefore it requires the system to work correctly from the beginning. The disadvantage of this method is high risk of failure. However, it can be used for small systems.

- **Advantages**
    - Less use of resources
- **Disadvantages**
    - High risk of failure

Pilot Approach

In this method the conversion is done in small parts. The advantage is that the effects of the new system applies only to a small part of the organization.

Phase in Approach

In this method the old system is gradually replaced by the new system. In the pilot approach organization is segmented, but in the phase in approach the computer system is segmented. The advantage of this system is the modular way of introducing the computer system.

Configuration Management Tools

1. GitHub

It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features.

It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

2. Sub Version

Apace Subversion is the most popular open source Configuration Management tool used for version controlling.

SVN has automatic merging and conflict resolution which makes locking unnecessary. Full revision history is maintained by for the files that are renamed, copied or moved. to mid projects.

3. Azure DevOps

Azure DevOps Server (previously known as Team Foundation Server).

Collaborative software development tools for the entire team to manage product lifecycle, reduce risks, and improve team efficiency. Comes with Cloud Based.

4. IBM Rational

It provides controlled access to software assets, including code, requirements, design documents, models, test plans and test results.

It features parallel development support, automated workspace management, baseline management, secure version management, reliable build auditing, and flexible access virtually anytime, anywhere.