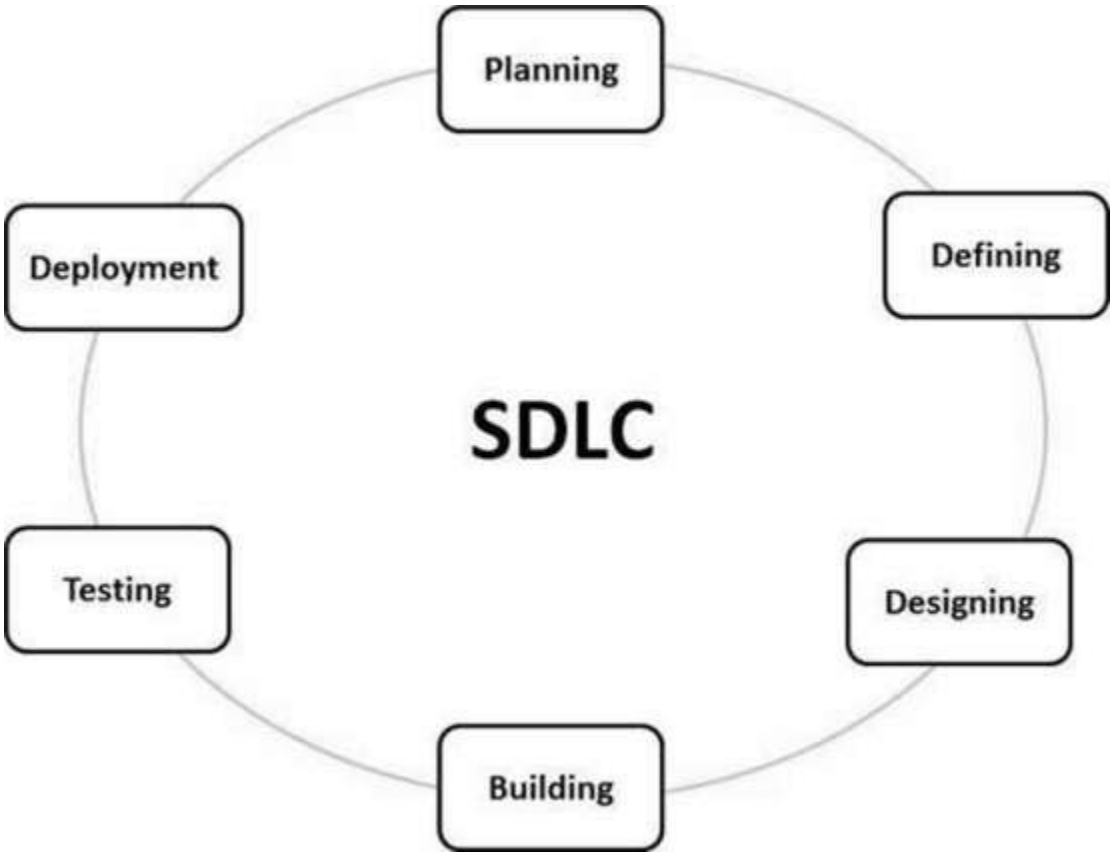
Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any)..

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle. 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.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

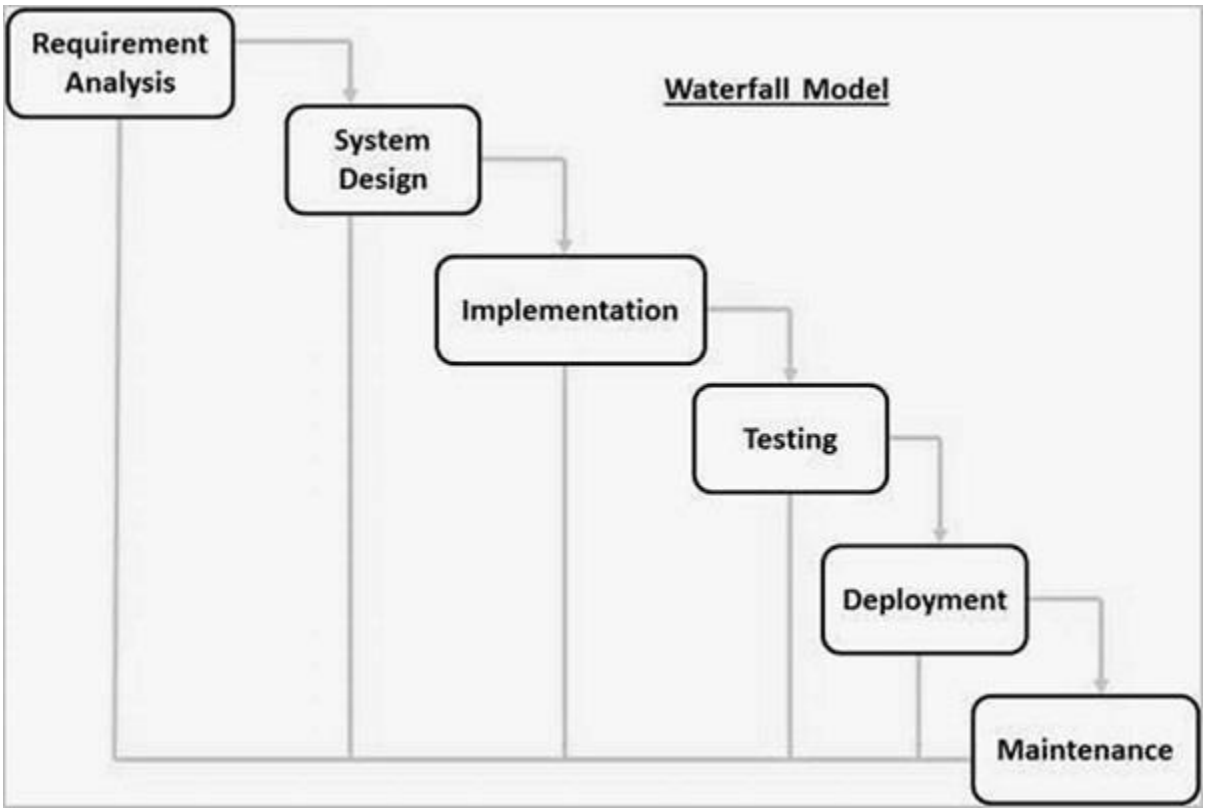
There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model
- Prototype model

Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.The Waterfall model is the earliest SDLC approach that was used for software development.The following illustration is a representation of the different phases of the Waterfall Model.



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.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

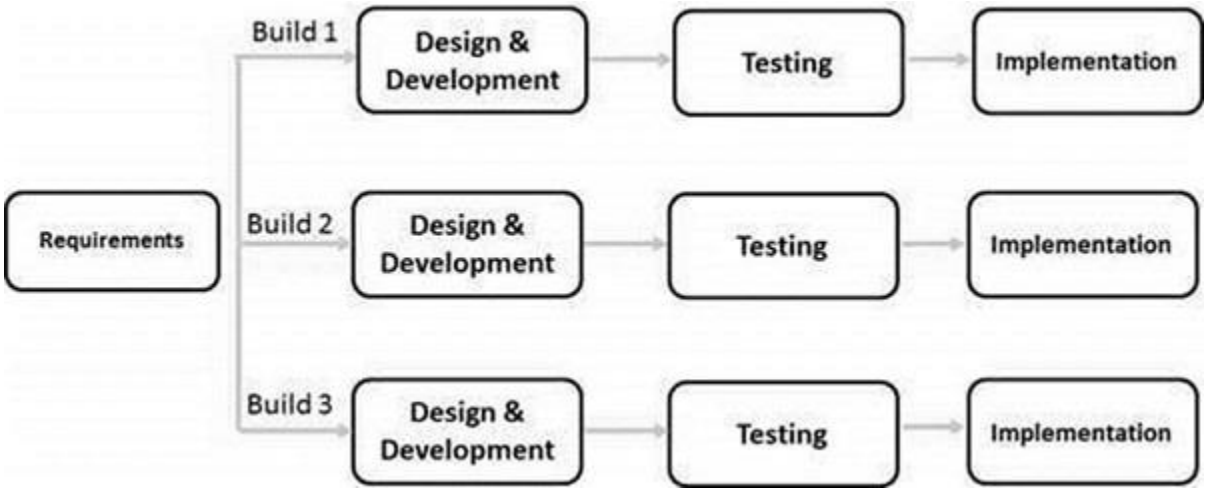
The major disadvantages of the Waterfall Model are as follows −

- 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 at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

Iterative Model

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

The key to a successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests must be repeated and extended to verify each version of the software.



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.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

The advantages of the Iterative and Incremental SDLC Model are as follows −

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.
- Easier to manage risk - High risk part is done first.

The disadvantages of the Iterative and Incremental SDLC Model are as follows −

- 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.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.

Spiral Model

This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

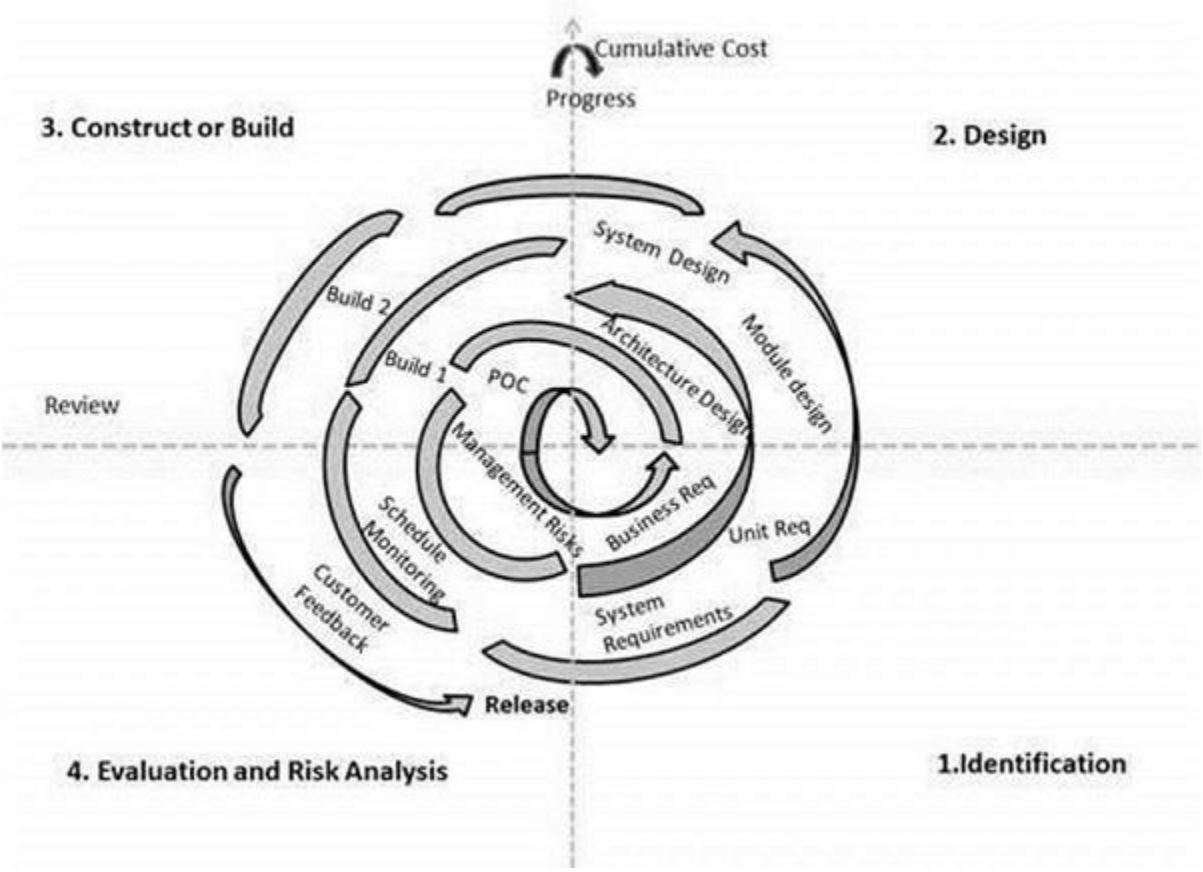
The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build

The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.



The following scenario explain the typical uses of a Spiral Model −

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- 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.
- Significant changes are expected in the product during the development cycle.

The advantages of the Spiral SDLC Model are as follows −

- 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.

The disadvantages of the Spiral SDLC Model are as follows −

- 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
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

Prototype Model

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Following is a stepwise approach explained to design a software prototype.

1.Basic Requirement Identification

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

2.Developing the initial Prototype

The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed. While, the workarounds are used to give the same look and feel to the customer in the prototype developed.

3.Review of the Prototype

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

4.Revise and Enhance the Prototype

The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like – time and budget constraints and technical feasibility of the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

Scenario where prototyping is useful

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.

The advantages of the Prototyping Model are as follows −

- 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.

The Disadvantages of the Prototyping Model are as follows −

- 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.

Feasibility study

A feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment.

Types of Feasibility study

1. Technical Feasibility

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system..

2. Economic Feasibility

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated.

3. Legal Feasibility

This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws, data protection acts or social media laws. Let’s say an organization wants to construct a new office building in a specific location. A feasibility study might reveal the organization’s ideal location isn’t zoned for that type of business. That

organization has just saved considerable time and effort by learning that their project was not feasible right from the beginning.

4. Operational Feasibility

This assessment involves undertaking a study to analyze and determine whether—and how well—the organization's needs can be met by completing the project. Operational feasibility studies also examine how a project plan satisfies the requirements identified in the requirements analysis phase of system development.

5. Scheduling Feasibility

This assessment is the most important for project success; after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete.

Below are some key benefits of conducting a feasibility study:

- Improves project teams' focus
- Identifies new opportunities
- Provides valuable information for a "go/no-go" decision
- Narrows the business alternatives
- Identifies a valid reason to undertake the project
- Enhances the success rate by evaluating multiple parameters
- Aids decision-making on the project
- Identifies reasons not to proceed

Systems Analysts

Systems analysts analyse how well software, hardware and the wider IT system fit the business needs of their employer or of a client. They write requirements for new systems and may also help implement them and monitor their effectiveness.

Typical responsibilities of the job include:

- examining current systems
- talking to users (requirements gathering)
- producing specifications for new or modified systems
- liaising with other IT staff such as programmers to produce new systems
- implementing new systems
- They are also responsible for user training and feedback.

Key skills for systems analysts

- Strong analytical skills
- Attention to detail
- Teamwork skills
- Written and verbal communication skills
- Interpersonal skills
- Flexibility
- Adaptability

Data flow Diagram (DFD)

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

DFD has often been used due to the following reasons:

- Logical information flow of the system
- Determination of physical system construction requirements
- Simplicity of notation
- Establishment of manual and automated systems requirements

DFD Symbols

There are **four basic symbols** that are used to represent a data-flow diagram.

Process

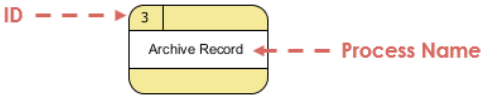
A process receives input data and produces output with a different content or form. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

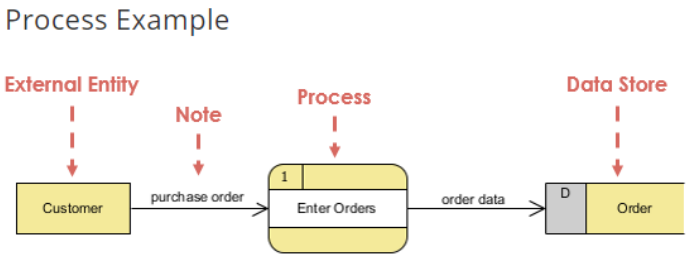
Example:

- Apply Payment
- Calculate Commission
- Verify Order

Notation

- A rounded rectangle represents a process
- Processes are given IDs for easy referencing





Data Flow

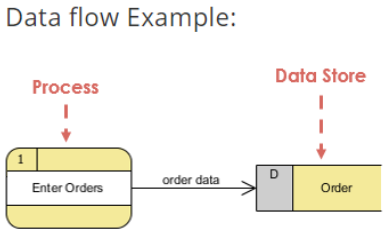
A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or it can represent a set of data element (or a data structure).

Example:

- Customer_info (LastName, FirstName, Tel #, etc.)
- Order_info (OrderId, OrderDate, CustomerID, etc.).

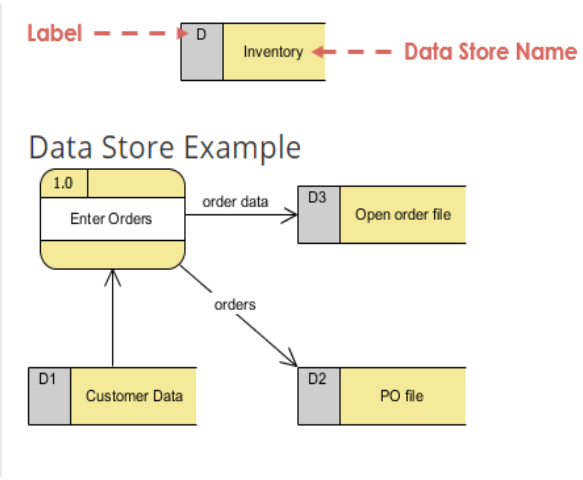
Notation

- Straight lines with incoming arrows are input data flow
- Straight lines with outgoing arrows are output data flows



Data Store

A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data because one or more processes need to use the stored data in a later time.



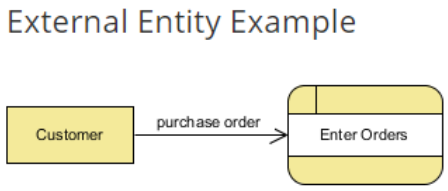
Notation

- Data can be written into the data store, which is depicted by an outgoing arrow
- Data can be read from a data store, which is depicted by an incoming arrow.
- Examples are: inventory, Accounts receivables, Orders, and Daily Payments.

External Entity

An external entity is a person, department, outside organization, or other information system that provides data to the system or receives outputs from the system. External entities are components outside of the boundaries of the information systems. They represent how the information system interacts with the outside world.

- A rectangle represents an external entity
- They either supply data or receive data
- They do not process data

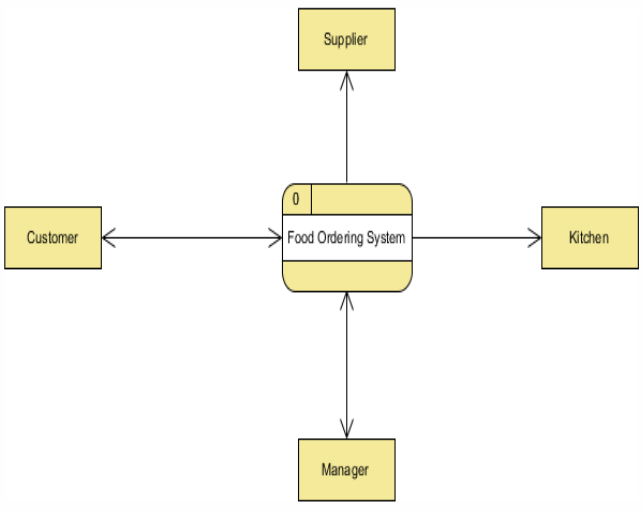


DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond.

Context Diagram

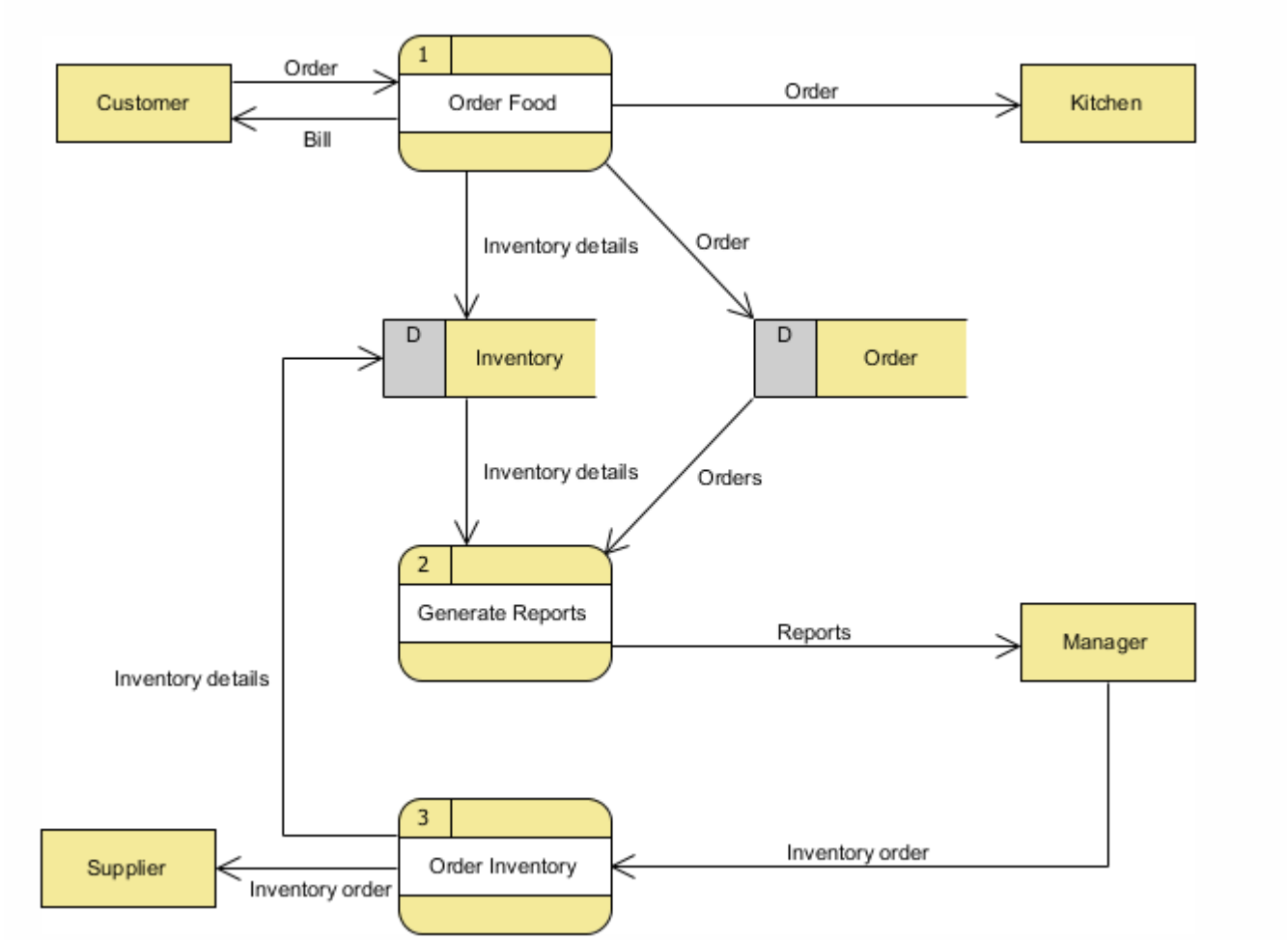
DFD Level 0 is also called a Context Diagram.

A context diagram gives an overview and it is the highest level in a data flow diagram, containing only one process representing the entire system. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.



Level 1 DFD

Processes in diagram 0 (with a whole number) can be exploded further to represent details of the processing activities. Example below shows the next level ((Diagram 1) of process explosion.



Decision Table

A decision table is a table that indicates conditions and actions in a simplified and orderly manner. In a decision table, the logic is well divided into conditions, actions (decisions) and rules for representing the various components that form the logical model. The general format of a decision table has four basic parts. They include:

- Action entry: It indicates the actions to be taken.
- Condition entry: It indicates conditions which are being met or answers the questions in the condition stub.
- Action stub: It lists statements that describe all actions that can be taken.
- Condition stub: it lists all conditions to be tested for factors necessary for taking a decision.

Advantages of Decision Table

- When the conditions are many then the decision table helps to visualize the outcomes of a situation.
- They are simple to understand and everyone can use this method to design the test scenarios and test cases.
- They are easy to draw.
- Decision tables can be changed easily according to the situation.
- Decision tables summarize all the outcomes of a situation and suggest suitable actions.

Disadvantages of Decision Tables

- Decision tables cannot express the complete sequence of operations to solve a problem; it may be difficult for a programmer to translate a decision table directly into a computer program.
- Decision tables do not show the flow of logic for the solution to a given problem.
- When there are many alternatives, decision table cannot list them all.
- Decision tables only present a partial solution.

Decision Tree

A decision tree is a decision support tool that uses a branching method to illustrate every possible outcome of a decision. A decision tree typically begins with a single node, which branches into possible outcomes. Each of those outcomes results to additional nodes, which branch off into other possibilities. This gives it a tree-like shape.

Advantages of Decision Trees

- Decision trees provide a clear indication of which fields are most important for prediction or classification.
- The tree output is easy to read and interpret.
- Decision trees are able to generate understandable rules.
- Decision trees require relatively little effort from users for data preparation.
- Decision trees perform classification without requiring much computation.
- Decision trees can handle both numerical and categorical variables.

Disadvantages of Decision Trees

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- They are unstable, in the sense that a small change in the structure of the optimal decision tree.
- They are relatively inaccurate when compared to other predictors. Many other predictors perform better with similar data.
- Calculations can get very complex, particularly if many values are uncertain if many outcomes are linked.