- **Learning Objectives**

At the end of the lesson, the learners should be able to:

- Identified the different hardware components.
- Identified the different software classes.
- Familiarized different emerging hardware and software technologies.

- What is Computer Hardware?
- Computer hardware or simply known as hardware Is the set of physical parts or any computer components that can be touch. Examples are Monitor, mouse, speaker, etc.
- Most Important Hardware Components
- **Motherboard :**It is a printed circuit board (PCB) and foundation of a computer that is the most extensive board in a computer chassis. It allocates power and allows communication to and between the other essential components of a computer, such as the CPU, RAM, and all other hardware components.

- What is Computer Hardware?
- **Central Processing Unit (CPU)**- CPUs reside in almost all devices you own, whether it's a smartwatch, a computer, or a thermostat. They are responsible for processing and executing instructions and act as the brains of your devices.
- What is Computer Hardware?

Three Major parts of the CPU

- **Control Unit (C.U.)-** The CPU part that sequentially accesses program instructions, decodes them, and coordinates the flow of data in and out of the ALU, registers, primary storage, and even secondary storage various output devices.
- **Arithmetic Logic Unit-** The part of the CPU that performs mathematical calculations and makes logical comparisons.
- **Register-** A high-speed storage area in the CPU used to temporarily hold small units of computer-generated instructions and data immediately before, during, and after execution by the CPU.
- What is Computer Hardware?
- **Memory-** A device used to store data or programs (sequences of instructions) on a temporary or permanent basis for use in an electronic digital computer.

- What is Computer Hardware?

Two types of Memory

- **Random Access Memory (RAM)-** also known as Main Memory or Primary Storage, is used to hold instructions and data while they are being used. RAM is volatile, meaning its contents are lost when the power goes off. RAM is more than 1000x faster than the fastest secondary storage.
- **Read Only Memory (ROM)-** non-volatile Memory that generally contains instructions for "booting" the computer (i.e. loading the operating system when the computer starts up).
- What is Computer Hardware?
- **Storage-** Used to store instructions and data while they are not being used. Example are hard disk, flash drive, memory cards, etc.
- What is Computer Hardware?
- **Input/Output Devices-** Alternatively referred to as an I.O. device, an input/output device is any hardware used by a human operator or other systems to communicate with a computer. As the name suggests, input/output devices can send data (output) to a computer and receive data from a computer (input).
    - Examples of input devices are keyboards, mouse, microphones, joystick, etc.
    - Example of output devices is Monitor, printers, speakers, etc.
- What is Computer Hardware?
- **Networking Devices**- Hardware devices connect computers, printers, fax machines, and other electronic devices to a network. These devices transfer data fast, secure, and correct way over the same or different networks.
- What is Computer Hardware?

WHAT DO YOU THINK?

Do we need to consider the hardware/hardware configuration when developing an application?

What kind of software you are interested to develop in the future?

Do we need to consider the software/software configuration when developing an application?

- Software

Software is a set of instructions, data, or programs that tell the computer how to work. Business, education, social division, and other fields can be applied in software designed to meet specific goals such as data manipulation, sharing of information, and others. It classified depends on the range of the application.

- Two Major Classes of Software
- **System software-** A set of programs to control the internal operations such as reading data from input devices, giving results to output devices, and ensuring proper functioning of components is called system software. System software includes the following:
  - Operating System
  - Development Tools, Programming Language software and Language processors
  - Device Drivers
  - Firmware
- Application software
- **Application software**- programs designed by the user to perform a specific function, such as accounting software, payroll software, etc.
  - **Real-Time Software**- A real-time system ensures an answer to an outside event within a specified period. The software that examines, evaluates and commands real-world events as they take place. Some example of real-time application is Video conferencing app like Google Meet and Zoom.
  - **Business Software**- This is software that holds relevant data from a database in which users can access to be used in a specific transaction such as payroll, accounting, inventory, and the like. This software facilitates a cost-effective structure in business operations and management decisions.

- Application software

- **Engineering and Scientific Software-** This software is meant to execute accurate computations on complex statistical data attained during a real-time environment. Applications heavily depend on engineering and scientific software such as the study of celestial bodies, under surface activities, and programming of an orbital path for shuttles.

- **Artificial Intelligence Software**- Software that is competent in "rational behavior." Developing this classification of software involves simulating a number of capabilities, including logic, discovering, problem-solving, assessment, information representation. Most of the emerging software nowadays has a touch of artificial intelligence.

▶ Application software

- **Web-Based Software**- It is a software that runs on a server (computer connected to the Internet), while users connect to it from their computers using an Internet browser. Any application that you can use through browser could be considered as web-based software.

- **Personal Computer (Pc) Software**- This class of software is used for both formal and personal use. This software is used predominantly in almost every field, whether home, school, business, and the like. It has emerged as a versatile tool for routine applications.

▶ Software

THINK ABOUT!

What kind of software you are interested to develop in the future?

▶ Assignment

Research the use and implication of the following emerging technologies in the application development. Identify also the hardware and software requirement for the following technologies.

- Internet of Things
- Cloud Computing
- Blockchain Technology
- Artificial Intelligence and Machine Learning
- Virtual Reality/ Augmented Reality
- 5G technology

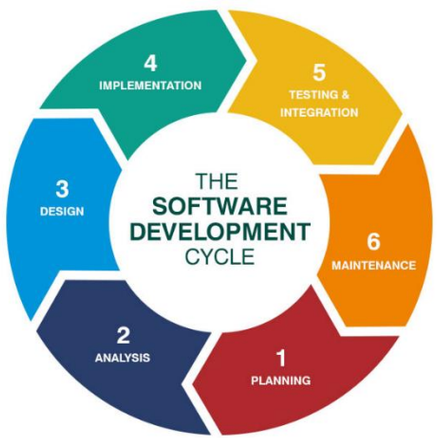
SDLC stands for Software Development Life Cycle. SDLC is a process that consists of a series of planned activities to develop or alter the Software Products. This tutorial will give you an overview of the SDLC basics, SDLC models available, and their application in the industry.

Software Development Life Cycle

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

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.

Software Development Life Cycle



Software Development Life Cycle

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 a product feasibility study in the economical, operational and technical areas.

Step 1: Planning and Requirements Analysis
Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage.
The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Step 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 by 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.

Step 3: Designing and 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.***
This **DDS** is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget, and time constraints, the best design approach is selected for the product.

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).
The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in **DDS**.

Step 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.
Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high-level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen concerning the type of software being developed.

Step 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**.

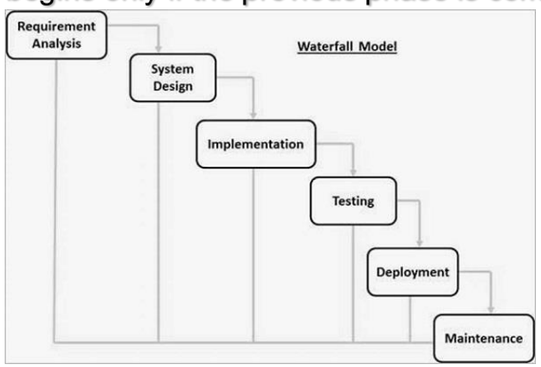
Step 6: Deployment in the Market and Maintenance
Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment 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 to 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.

SDLC Models:

- Waterfall Model
- Iterative Model
- Spiral Model
- V Model
- Big bang 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 *waterfall Model* illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.



Waterfall approach was first to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Waterfall Model – Design

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model – Design

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model – Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. 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

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

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.

Waterfall Model – Disadvantages

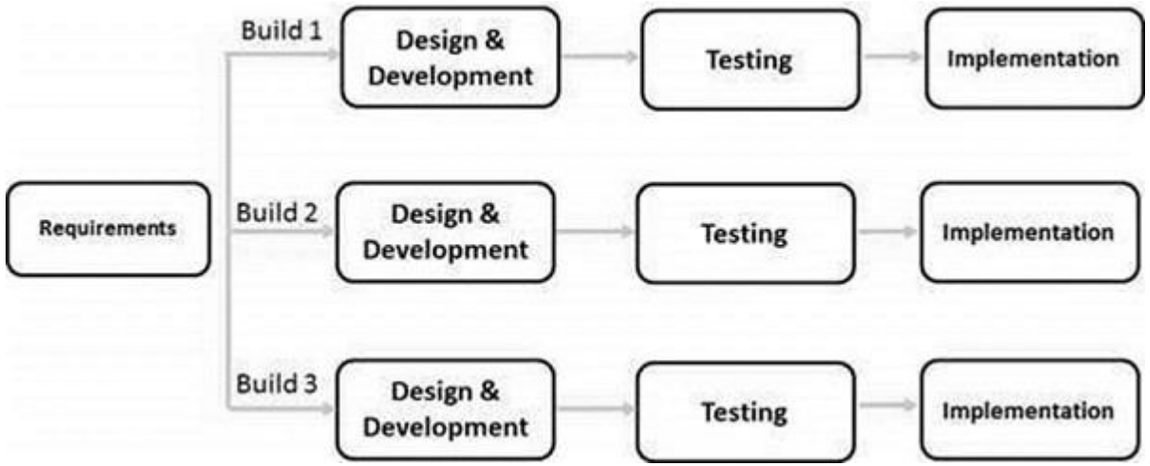
The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

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.
- ➢ Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Model – Design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).



Iterative Model – Design

- Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach.“
- In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.
- 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.

Iterative Model – Application

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. 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.

Iterative Model – Pros and Cons

The advantage of this model is that there is a **working model of the system at a very early stage of development**, which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

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.

Iterative Model – Design

- 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.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment.
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.

- During the life cycle, software is produced early which facilitates customer evaluation and feedback.

Iterative Model – Design

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

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.

Iterative Model – Design

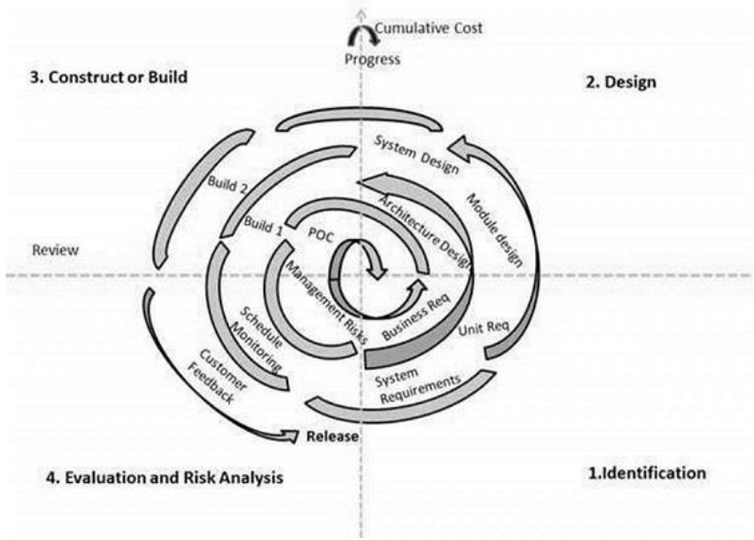
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which is a risk.
- Highly skilled resources are required for risk analysis.
- Projects progress is highly dependent upon the risk analysis phase.

SPIRAL MODEL

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall 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. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

SPIRAL MODEL – DESIGN

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.



SPIRAL MODEL – DESIGN

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.

SPIRAL MODEL

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.

SPIRAL MODEL

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.
Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

SPIRAL MODEL
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.

SPIRAL MODEL
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.

SPIRAL MODEL APPLICATION
The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.
The following pointers 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.

SPIRAL MODEL

- 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

SPIRAL MODEL PROS & CONS

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity.

Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

SPIRAL MODEL PROS & CONS

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.

SPIRAL MODEL PROS & CONS

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.

V MODEL

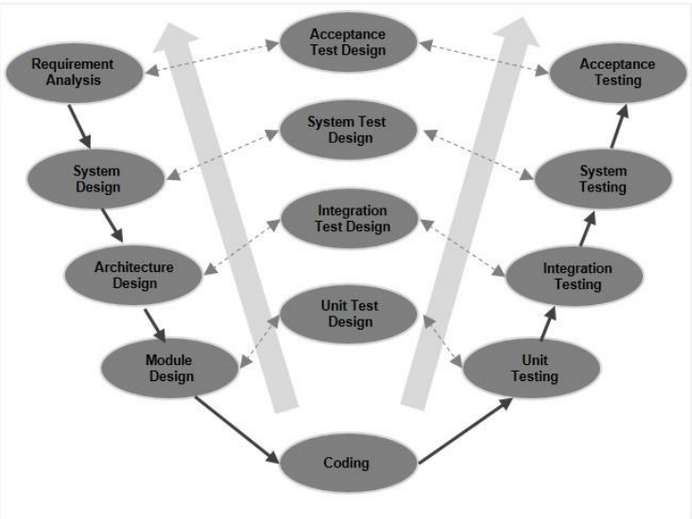
The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

V MODEL

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

V MODEL



V MODEL VERIFICATION PHASES

There are several Verification phases in the V-Model, each of these are explained in detail below.

Business Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing

V MODEL VERIFICATION PHASES

System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

V MODEL VERIFICATION PHASES

Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

V MODEL VERIFICATION PHASES

Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

V MODEL VERIFICATION PHASES

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

V MODEL VALIDATION PHASES

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

V MODEL VERIFICATION PHASES

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

V MODEL APPLICATION

V- Model application is almost the same as the waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly a disciplined domain.

V MODEL APPLICATION

The following pointers 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 PROS & CONS

The advantage of the V-Model method is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today's dynamic world, it becomes very expensive to make the change.

V MODEL PROS & CONS

The advantages of the V-Model method are as follows:

- 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 due to the rigidity of the model. Each phase has specific deliverables and a review process.

V MODEL PROS & CONS

The disadvantages of the V-Model method are as follows −

- High 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.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

BIGBANG MODEL

The Big Bang model is an SDLC model where we do not follow any specific process. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement. This Big Bang Model does not follow a process/procedure and there is a very little planning required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

Big Bang Model – Design and Application

The Big Bang Model comprises of focusing all the possible resources in the software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given.

BIGBANG MODEL Pros and Cons

The advantage of this Big Bang Model is that it is very simple and requires very little or no planning. Easy to manage and no formal procedure are required. However, the Big Bang Model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

BIGBANG MODEL Pros and Cons

The advantages of the Big Bang Model are as follows:

- This is a very simple model
- Little or no planning required
- Easy to manage
- Very few resources required
- Gives flexibility to developers
- It is a good learning aid for new comers or students.

BIGBANG MODEL Pros and Cons

The disadvantages of the Big Bang Model are as follows:

- Very High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Can turn out to be very expensive if requirements are misunderstood.

AGILE METHODOLOGY

Agile Methodology- Agile model is a combination of iterative and incremental process models with a focus on process adaptability and customer satisfaction by rapid delivery of working software products. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.

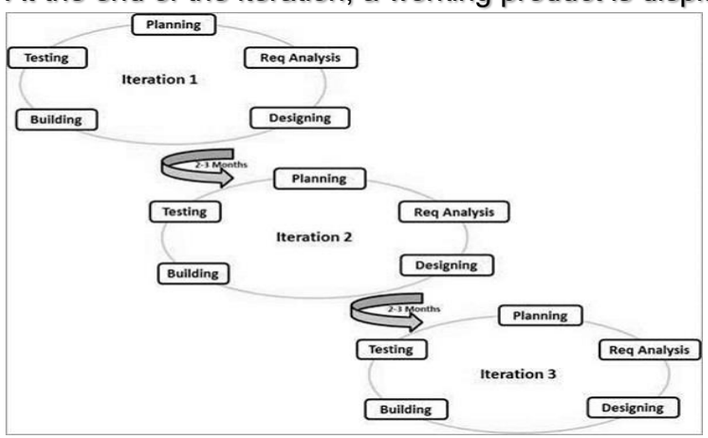
AGILE METHODOLOGY

Every iteration involves cross-functional teams working simultaneously on various areas like:

1. Planning
2. Requirements Analysis
3. Design
4. Coding
5. Unit Testing and
6. Acceptance Testing.

AGILE METHODOLOGY

At the end of the iteration, a working product is displayed to the customer and critical stakeholders.



AGILE METHODOLOGY

Agile testing methods:

a. Scrum

b. Crystal
c. Dynamic Software Development Method(DSDM)
d. Feature Driven Development(FDD)
e. Lean Software Development
f. eXtreme Programming(XP)

Rapid Application Development (RAD)

Rapid Application Development (RAD)- Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

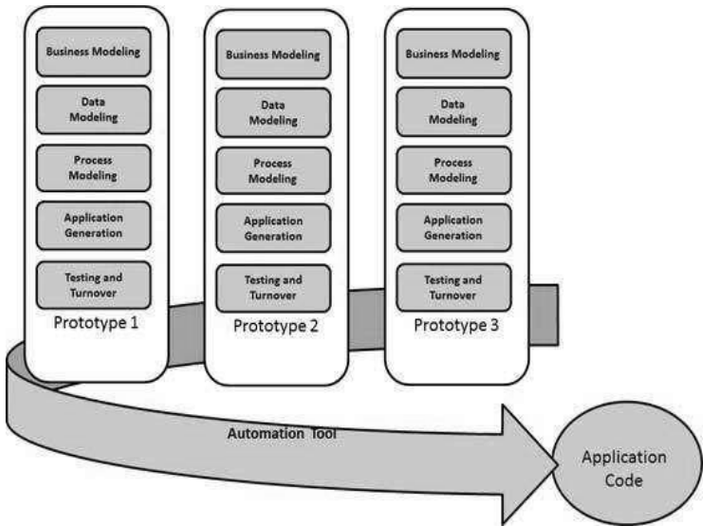
In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

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

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

RAD

Don’t go chasing waterfalls—get used to the rapids. Explore rapid application development for faster software delivery and continuous iterations.



RAD

Rapid Application Development (RAD) is a form of agile software development methodology that prioritizes rapid prototype releases and iterations. Unlike the Waterfall method, RAD emphasizes the use of software and user feedback over strict planning and requirements recording.

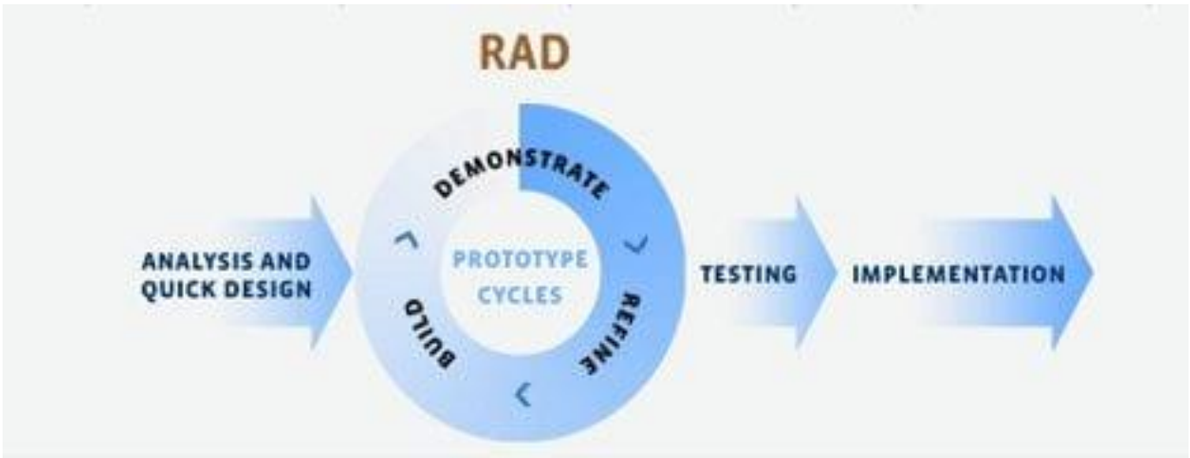
RAD

Some of the key benefits and advantages of RAD are:

- Enhanced flexibility and adaptability as developers can make adjustments quickly during the development process.
- Quick iterations that reduce development time and speed up delivery.
- Encouragement of code reuse, which means less manual coding, less room for errors, and shorter testing times.
- Increased customer satisfaction due to high-level collaboration and coordination between stakeholders (developers, clients, and end users).
- Better risk management as stakeholders can discuss and address code vulnerabilities while keeping development processes going.
- Fewer surprises as, unlike the Waterfall method, RAD includes integrations early on in the software development process.

RAD

5 steps or phases in RAD



RAD

Step 1. Define and finalize project requirements

During this step, stakeholders sit together to define and finalize project requirements such as project goals, expectations, timelines, and budget. When you have clearly defined and scoped out each aspect of the project's requirements, you can seek management approvals.

RAD

Step 2: Begin building prototypes

As soon as you finish scoping the project, you can begin development. Designers and developers will work closely with clients to create and improve upon working prototypes until the final product is ready.

RAD

Step 3: Gather user feedback

In this step, prototypes and beta systems are converted into working models. Developers then gather feedback from users to tweak and improve prototypes and create the best possible product.

RAD

Step 4: Test, test, test

This step requires you to test your software product and ensure that all its moving parts work together as per client expectations. Continue incorporating client feedback as the code is tested and retested for its smooth functioning.

RAD

Step 5: Present your system

This is the final step before the finished product goes to launch. It involves data conversion and user training.

RAD

Is your team RAD-ready?

Here's a checklist that will help you determine your team's RAD-readiness:

- ✓ **Do you need to develop a software product within a short span of time (two to three months)?**
- ✓ **Do you have an experienced team of developers, coders, and designers who can carry out the work on your timeline?**
- ✓ **Is your client open to adopting RAD, i.e., will the client be available for collaboration throughout the software development process?**
- ✓ **Do you have the right tools and technology to implement RAD?**

RAD

Adopting a new process requires buy-in from everyone involved, including your team and your client. If you've decided the RAD approach is right for you, here's what you should do next:

RAD

1. Make sure your team has an all-hands-on-deck mindset. Talk to your team about the benefits of the new approach and listen to and address their concerns.
2. Ensure that all stakeholders are willing to adhere to the project timelines.
3. Explore application development software and tools. Invest in one that fits your business's budget and requirements to be able to effectively apply this methodology.

MODELS & METHODOLOGIES

Factors Need to Consider in Choosing Software Development Method for your System

- **Determine the Level of Flexibility in the Requirements-** You need to consider the flexibility of your specification before choosing an SDLC model. The Agile and Iterative methods are ideal for a web and app development in which changes are frequently introduced along the line. The waterfall is ideal for a classic web and app development where stability and predictability in the various phases of the development are preeminent.

If you're targeting a controlled group of end-users, you'll most likely have a mostly fixed set of requirements to work with, and that would make the waterfall method ideal for your web and app development. But if your target end users are dispersed, you'll most likely have to deal with wads of feedback after the app launch requesting the inclusion of new features, so the Agile or iteration methods would be the best SDLC models in this case.

MODELS & METHODOLOGIES

Factors Need to Consider in Choosing Software Development Method for your System

- **Scale and Scope of the Development-** The scale of a project determines the number of developers needed to handle it. The larger the project, the larger the size of the development team. Larger projects, therefore, require much more elaborate and orderly project management plans, and in such cases, the good old waterfall module is the best-suited.

MODELS & METHODOLOGIES

Factors Need to Consider in Choosing Software Development Method for your System

- **Determine Whether a Sprint or Marathon Time-Frame is Best-Suited-** For developments that unfold through sprints, the Agile and iterative methods are the best, as they facilitate the release of partially completed systems to generate an impression of rapid progression. But if the time-frame for the development is long-term and there are no fast-approaching deadlines, the waterfall method is a great choice.

MODELS & METHODOLOGIES

Factors Need to Consider in Choosing Software Development Method for your System

- **Consider Your Developer Team's Location-** If your developer team is dispersed across the map, then there'll be a greater need for coordination, coherence and accountability. In this case, a more rigid project management regimen is the best-suited, and this is the case where the waterfall shines the most. Agile requires much more frequent contact and closely-knit teams; a dispersed developer team might have to deal with lots of confusion and missteps in the development process if Agile is the chosen SDLC.

ITC311
Application Development and Emerging Technologies

UNIT 4. Software Design Principles and Design Patterns

A. General Learning Outcome:

Discuss Software Design Principles and Design Patterns. Specific Learning Outcomes:

1. Discuss the different Software Design Principles
2. Discuss different Design Patterns.
3. Differentiate Software Design Principles and Design Patterns.

B. Pre-Test

1. What is the importance of Prototyping process?
2. What are the different types of prototyping processes?
3. What are quality assurance principles?
4. Why is there ISO 9126 and ISO 25010?

C. Content

SOFTWARE DESIGN PRINCIPLES

SOFTWARE DESIGN PRINCIPLES

Principles of design are the laws of designing anything. In other words, to have a good design, you should consider these principles for the best design possibleA design is said to be good design if

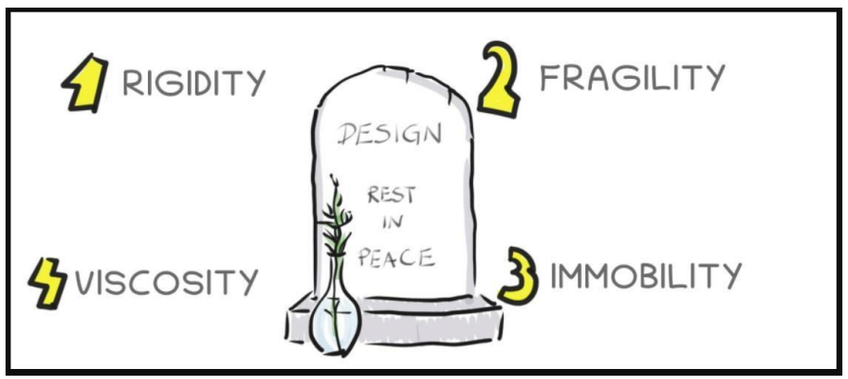
- ✓ Ensuring that we actually conform with the requirements
- ✓ Increasing qualities such as: Usability, Efficiency, Reliability,
- ✓ Maintainability, Reusability , etc
- ✓ Increasing profit by reducing cost and increasing revenue
- ✓ Accelerating development

In order to achieve the design goals we need to follow design principles mentioned in the following lectures. The principles are especially applicable for object oriented systems

1. Divide and conquer
2. Increase cohesion where possible
3. Decrease coupling where possible
4. Keep the level of abstraction as high as possible

5. Increase reusability where possible
6. Reuse existing designs and code where possible
7. Design for flexibility
8. Anticipate obsolesce
9. Design for Portability
10. Design for Testability
11. Design defensively

ROTTING DESIGN



The design of many software applications begins as a vital image in the minds of its designers. At this stage it is clean, elegant, and compelling. Some of these applications manage to maintain this purity of design through the initial development and into the first release.

But then something begins to happen.The software starts to rot. At first it isn't so bad. However, gradually the developers find increasingly hard to maintain the software. Eventually the effort required to make even the simplest of changes to the application becomes so high that the engineers and front line managers cry for a redesign project.

However, such redesigns rarely succeed. Because though the designers start out with good intentions, they find that they are shooting at a moving target. The old system continues to evolve and change. Because of this continues change design starts rotting.

Symptoms of Rotting Design

There are **four primary symptoms** that tell us that our designs are rotting. They are not orthogonal, but are related to each other in ways that will become obvious. they are:

- Rigidity
- Fragility
- Immobility
- Viscosity

Rigidity is the tendency for software to be difficult to change, even in simple ways. Every change causes a cascade of subsequent changes in dependent modules. What begins as a simple two day change to one module grows into a multi-week marathon of change in module after module as the engineers chase the thread of the change through the application. When software behaves this way, managers fear to allow engineers to fix non- critical problems. When the manager's fears become so acute that they refuse to allow changes to software, official rigidity sets in.

Fragility is the tendency of the software to break in many places every time it is changed. Often the breakage occurs in areas that have no conceptual relationship with the area that was changed. Such errors fills the hearts of managers with foreboding. Every time they authorize a fix, they fear that the software will break in some unexpected way. As the fragility becomes worse, the probability of breakage increases with time, asymptotically approaching 1. Such software is impossible to maintain. Every fix makes it worse, introducing more problems than are solved.

Immobility is the inability to reuse software from otherprojects or from parts of the same project. It often happens that one engineer will discover that he needs a module that is similar to one that another engineer wrote. However, it also often happens that the module in question has too much baggage that it depends upon. After much work, the engineers discover that the work and risk required to separate the desirable parts of the software from the undesirable parts are too great to tolerate. And so the software is simply rewritten instead of reused but solved.

First, fluids with high viscosity, like peanut butter, don't flow as smoothly as low-viscosity fluids like water. ***Viscosity*** of design comes in two forms: of the design, and of the environment. When faced with a change, engineers usually find more than one way to make the change. Some of the ways preserve the design, others do not (i.e. they are hacks.) When the design preserving methods are harder to employ than the hacks, then the viscosity of the design is high. *It is easy to do the wrong thing, but hard to do the right thing.* Viscosity of environment comes about when the development environment is slow and inefficient. E.g. If compile times are very long, engineers will be tempted to make changes that don't force large recompiles, even though those changes are not optimal from a design point of view.

The four symptoms are either directly, or indirectly caused by *improper* ***dependencies*** between the modules of the software. It is the dependency architecture that is degrading, and with it the ability of the software to be maintained. So, the dependencies between modules in an application must be managed This management consists of the creation of dependency firewalls - Across such firewalls, dependencies do not propagate. OOD is replete with principles and techniques/design patterns for building such firewalls, and for managing module dependencies. Design principles collectively called SOLID by Robert C Martin are grouped as Design Principles= II (for this course) & discussed in next slides

Software design principles are a set of guidelines that helps developers to make a good system design. The most important principle is SOLID principle. The key software design principles are as:

S.O.L.I.D

It is combination of five basic designing principles.

Single Responsibility Principle (SRP)

This principle states that there should never be more than one reason for a class to change. This means that you should design your classes in such a way that each class should have a single purpose.

Example - An Account class is responsible for managing Current and Saving Account but a CurrentAccount and a SavingAccount classes would be responsible for managing current and saving accounts respectively. Hence both are responsible for single purpose only. Hence we are moving towards specialization.

Class should have only one reason to change.

Related to and derived from cohesion, i.e. that elements in a module should be closely related in their function There should never be more than one reason for a class to change. Responsibility of a class to perform a certain function is one of the reasons for the class to change. Plan your classes based on what is likely to change. If there is more than one thing that may change, split the class until there is only one thing likely to change in each class. Each class will handle only one responsibility and on future if we need to make one change we are going to make it in the class which handle it.

Otherwise, when we need to make a change in a class having more responsibilities the change might affect the other functionality of the classes. Introduced by Tom DeMarco and reinterpreted the concept and defined the responsibility as a reason to change by Robert Martin.

Example:

```
Modem.java                        //SRP Violation interface Modem {
public void dial(String pno);     //PhoneLine public void hangup();
-----------------------------------------------
```

public void send(char c); // Connection public char recv(); }

Open/Closed Principle (OCP)

This principle states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. The "closed" part of the rule states that once a module has been developed and tested, the code should only be changed to correct bugs. The "open" part says that you should be able to extend existing code in order to introduce new functionality.

Example – A PaymentGateway base class contains all basic payment related properties and methods. This class can be extended by different PaymentGateway classes for different payment gateway vendors to achieve theirs functionalities. Hence it is open for extension but closed for modification.

Software entities like classes, modules and functions should be open for extension but closed for modifications. (Bertrand Meyer)

The OCP says that we should attempt to design modules that never need to be changed Modules that conform to the OCP meet two criteria

- Open For Extension - the behaviour of the module can be extended to meet new requirements
- Closed For Modification - the source code of the module is not allowed to change How can we do this?

  Using Abstraction, Polymorphism, Inheritance & Interfaces

It is not possible to have all the modules of a software system satisfy the OCP. But we should attempt to minimize the number of modules that do not satisfy it. Conformance to this principle yields the greatest level of reusability and maintainability. It is important for backward compatibility, regression testing

Example:

```
public double totalPrice(Part[] parts) {
  double total = 0.0;
  for (int i=0; i<parts.length; i++) {
    total += parts[i].getPrice();
  }
  return total;
}
```

If **Part** is a base class or an interface and polymorphism is bein used, then this class ca easily accommodate new types of parts without having to be modified!

Bellow is an example which violates the Open ClosePrinciple

Since the GraphicEditor class has to be modified for every new shape class that has to be added. There are several disadvantages. Adding new type of shape is time consuming as it is difficult to understand the logic of the GraphicEditor.

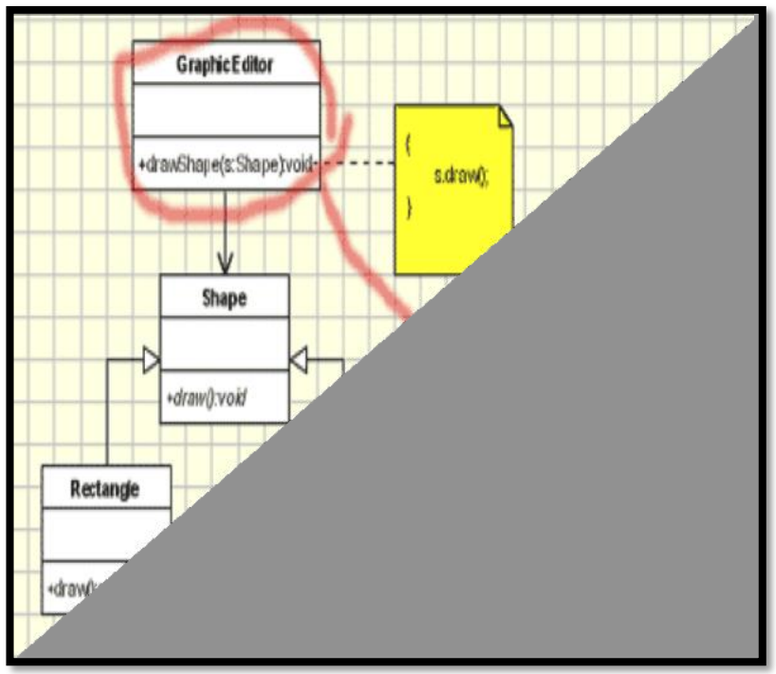
Open Close Principle: Example

each new shape added the unit testing of the GraphicEditor should be redone. adding a new shape might affect the existing functionality in an undesired way.

Adding new type of shape is time consuming as it is difficult to understand the logic of the GraphicEditor. The above example can be modified to support OCP as follows

Now no unit testing required. need to understand the sourcecode fromGraphicEditor.

since the drawing code is moved to the concrete shape classes, it's a reduced risk to affect old functionality



Liscov Substitution Principle (LSP)

This principle states that functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

Example - Assume that you have an inheritance hierarchy with Person and Student. Wherever you can use Person, you should also be able to use a Student, because Student is a subclass of Person.

Subclasses should be substitutable for their base classes OR Functions that use references to base (super) classes must be able to use objects of derived (sub) classes without knowing it.

LSP was introduced by Barbara Liskov in a 1987. The LSP seems obvious given all we know about polymorphism, but there are subtleties that need to be considered.

For example:

```
public void drawShape(Shape s) { // Code here. }
```

The drawShape method should work with any subclass of the Shape superclass or, if Shape is a Java interface, it should work with any class that implements the Shape interface.

But we must be careful when we implement subclasses to insure that we do not unintentionally violate the LSP. If a function does not satisfy the LSP, then it probably makes explicit reference to some or all of the subclasses of its super class. Such a function also violates the OCP, since it may have to be modified whenever a new subclass is created.

Example

Consider the Rectangle class

```
public class Rectangle {
  private double width;
  private double height;
  public Rectangle(double w, double
    width = w;
    height = h;
  }
  public double getW
  public double
  public void
  public
  pub
}
```

What about a Square class?

Clearly, a square is a rectangle, so the Square class should be derived from the Rectangle class, right? Let's see!

```
public class Square extends Rectangle {
  public Square(double s) {super(s, s);}
  public void setWidth(double w) {
    super.setWidth(w);
    super.setHeight(w);
  }
  public void setHeight(double h) {
    super.setHeight(h);
    super.setWidth(h);
  }
}
```

Observations:

A square does not need both a width and a height as attributes, but it will inherit them from Rectangle anyway. So, each Square object wastes a little memory, but this is not a major concern.

The inherited setWidth() and setHeight() methods are not really appropriate for a Square, since the width and height of a square are identical.

So we'll need to override setWidth() and setHeight(). This shows this might not even be an appropriate use of inheritance!

But check this out!

```
public class TestRectangle {
  public static void testLSP(Rectangle r) {
    r.setWidth(4.0); r.setHeight(5.0);
    System.out.println("Width is 4.0 and Height is 5.0" +
                       ", so Area is " + r.area());
    if (r.area() == 20.0) System.out.println("Looking good!\n");
    else
      System.out.println("Huh?? What kind of rectangle is this??\n");
  }
```

```
    public static void main(String args[]) {
    Rectangle r = new Rectangle(1.0, 1.0);
    Square s = new Square(1.0);
    testLSP(r);
    testLSP(s);
    }
  }
```

Test program output:

Width is 4.0 and Height is 5.0, so Area is 20.0 Looking good!

Width is 4.0 and Height is 5.0, so Area is 25.0 Huh?? What kind of rectangle is this?? Looks like we violated the LSP!. What's the problem here?

The programmer of the testLSP() method made the reasonable assumption that changing the width of a Rectangle leaves its height unchanged. Passing a Square object to such a method results in problems, exposing a violation of the LSP The Square and Rectangle classes look self consistent and valid.

Yet a programmer, making reasonable assumptions about the base class, can write a method that causes the design model to break down. Solutions can not be viewed in isolation, they must also be viewed in terms of reasonable assumptions that might be made by users of the design. A mathematical square might be a rectangle, but a Square object is not a Rectangle object; the behaviour of a Square object is not consistent with the behaviour of a Rectangle object!

Behaviourally, a Square is not a Rectangle! A Square object is not polymorphic with a Rectangle object.

Interface Segregation Principle (ISP)

This principle states that Clients should not be forced to depend upon interfaces that they don't use. This means the number of members in the interface that is visible to the dependent class should be minimized.

Example - The service interface that is exposed to the client should contains only client related methods not all.

Clients should not be forced to depend upon interfaces that they don't use or many client specific interfaces are better than one general purpose interface

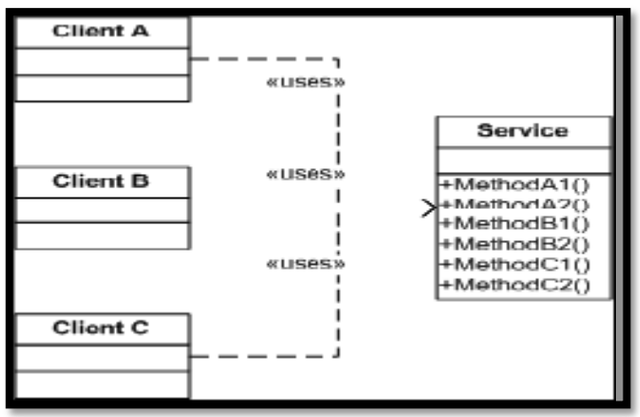
If we add methods that should not be there the classes implementing the interface will have to implement those methods as well.

For example if we create an interface called Worker and add a method lunch break, all the workers will have to implement it.

What if the worker is a robot?

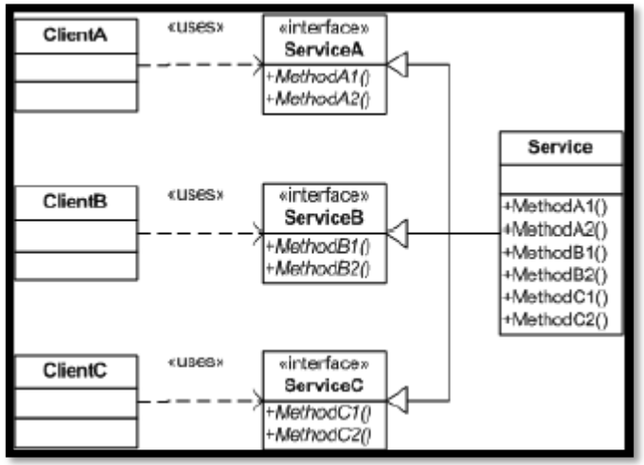
An interfaces containing methods that are not specific to it are called polluted or fat interfaces - we should avoid them.

Fat classes cause bizarre and harmful couplings between their clients. When one client forces a change on the fat class, all the other clients are affected. Thus, clients should have to depend only on methods that they call. This can be achieved by breaking the interface of the fat interface into many client-specific interfaces. Each client-specific interface declares only those functions that its particular client or client group invoke. The fat class can then inherit all the client-sTphisec ific interfaces and implement them.



The classes A, B and C are clients of Service class who defines methods to used by them. Each one of the client classes uses small and different port of these

methods.



Now, we create client specific interfaces, each interface is more granular and provides more cohesive definition for each client.

Dependency Inversion Principle (DIP)

The Dependency Inversion Principle states that:

- High level modules should not depend upon low level modules. Both should depend upon abstractions.
- Abstractions should not depend upon details. Details should depend upon abstractions.

It helps us to develop loosely couple code by ensuring that high-level modules depend on abstractions rather than concrete implementations of lower-level modules. The Dependency Injection pattern is an implementation of this principle

Example - The Dependency Injection pattern is an implementation of this principle Abstractions should not depend on details. Details should depend on abstractions.

If the high level module depends on the low level module, changes to the lower-level modules can have direct effects on the higher-level modules and can force them to change in turn. It becomes very difficult to reuse those high-level modules in different contexts.

Further, instead of writing our abstractions based on details, we should write the details based on abstractions.

- No variable should hold a reference to a concrete class
- No class should derive from a concrete class
- No method should override an implemented method of its base class
- if it is something that can be changed, separate it into its own class

However, there seems no reason to follow this heuristic for classes that are concrete but nonvolatile and most concrete classes that we write as part of an application program are volatile. It is those concrete classes that we do not want to depend directly on. Their volatility can be isolated by keeping them behind an abstract interface. DIP helps you make your design OCP compliant.

Dependency inversion can be applied wherever one class sends a message to another. For example, consider the case of the Button object and the Lamp object. On receiving the Poll message, the Button object determines whether a user has "pressed" it.

- Sensing mechanism can be any thing; GUI, physical button pressed, motion detector
- The Button object detects that a user has either activated or deactivated it.

The Lamp object either turns off or turns on. How can we design a system such that the Button object controls the Lamp object?

Naive model of a Button and a Lamp. Why is this naïve? Consider the following code: public class Button{
private Lamp lamp; public void Poll() {
if (/*some condition*/) lamp.TurnOn();
}
}

Note that the Button class depends directly on the Lamp class. This dependency implies that Button will be affected by changes to Lamp. Moreover, it will not be possible to reuse Button to control other objects (e.g. a Motor object)

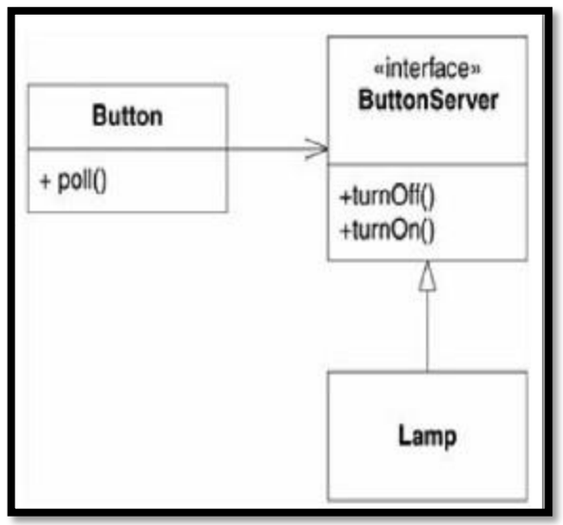
In this model, Button objects control Lamp objects and only Lamp objects. Hence, this solution violates DIP

The high-level policy of the application has not been separated from the low-level implementation. The abstractions have not been separated from the details. Without such a separation, the high-level policy automatically depends on the low-level modules, and the abstractions automatically depend on the details.

High-level policy is the abstraction that underlies the application, the truths that do not vary when the details are changed. In the Button/Lamp example, the underlying abstraction is to detect an on/off gesture from a user and relay that gesture to a target object.

What mechanism is used to detect the user gesture?Irrelevant! What is the target object? Irrelevant!

These are details that do not impact the abstraction. The previous model can be improved by inverting the dependency upon the Lamp object. The Button now holds an association to something called a ButtonServer, which provides the interfaces that Button can use to turn something on or off.



Lamp implements the ButtonServer interface. Thus, Lamp is now doing the depending rather than being depended on. This model allows a Button to control any device that is willing to implement the ButtonServer interface. This gives us a great deal of flexibility and this solution also puts a constraint on any object that needs to be controlled by a Button.

Such an object must implement the ButtonServer interface. These objects may also want to be controlled by a Switch object or some kind of object other than a Button.

DRY (Don’t Repeat Yourself)

This principle states that each small pieces of knowledge (code) may only occur exactly once in the entire system. This helps us to write scalable, maintainable and reusable code.

Example – Asp.Net MVC framework works on this principle.

KISS (Keep it simple, Stupid!)

This principle states that try to keep each small piece of software simple and unnecessary complexity should be avoided. This helps us to write easy maintainable code.

YAGNI (You ain't gonna need it)

This principle states that always implement things when you actually need them never implements things before you need them.

SOFTWARE DESIGN PATTERNS

Origins of Design Patterns

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

Christopher Alexander, A Pattern Language, 1977 Context: City Planning and Building architectures

Software design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Many of them have been systematically documented for all software developers to use. A good pattern should be as general as possible contain a solution that has been proven to effectively solve the problem in the indicated context.

Uses of Design Patterns

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.
`

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

Elements of Design Patterns

Design patterns must be described in an easy-to-understand form so that people can determine when and how to use it.

Minimum parameters to describe patterns are:

- ✓ **Name** - Important because it becomes part of a design vocabulary
- ✓ **Problem** - Describes intent, context, and when the pattern is applicable
- ✓ **Solution** - Design elements and their relationships.
- ✓ **Consequences** - Tradeoffs of applying the pattern
    - Each pattern has costs as well as benefits

- Issues include flexibility, extensibility, etc.

Each pattern in GoF is documented with following specifications:

Intent, also known as, motivation, applicability, structure, participants, collaborations, consequences, implementation, sample code, known uses, related patterns

Examples of Patterns

Design patterns you have already seen:

- Encapsulation (Data Hiding)
- Sub-classing (Inheritance)
- Iteration
- Exceptions

Encapsulation Pattern

Problem: Exposed fields are directly manipulated from outside, leading to undesirable dependences that prevent changing the implementation.

Solution: Hide some components, permitting only stylized access to the object.

Sub-classing Pattern

Problem: Similar abstractions have similar members (fields and methods). Repeating these is tedious, error- prone, and a maintenance headache.

Solution: Implementations perform traversals. The results are communicated to clients via a standard interface.

Exceptions Pattern

Problem: Code is cluttered with error-handling code.

Solution: Errors occurring in one part of the code should often be handled elsewhere. Use language structures for throwing and catching exceptions.

Why Patterns?

Speed up the development process by providing tested, proven development paradigms. Shared language of design

- Increases communication bandwidth because it has consistent documentation
- decreases misunderstandings Learn from experience

- Becoming a good designer is hard.
- Understanding good designs is a first step
- Tested solutions to common problems Used to achieve the following quality attributes;

- Modifiability
- Exchangeability
- Reusability
- Extensibility
- Maintainability
- Reliability
- Testability

Patterns help you to manage software complexity.

Types of Design Patterns

Design pattern are granular and applied at different levels such as frameworks, and subsystems GRASP Patterns

- General Responsibility Assignment Software Pattern
- E.g. Information Expert, Creator, Low Coupling, High Cohesion Architectural patterns

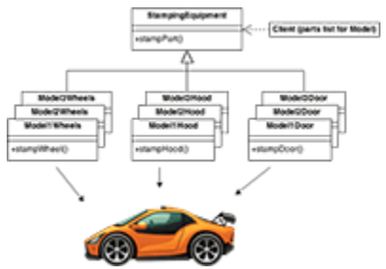
- An architectural pattern express a fondamental structural
- organisation schéma for software Systems

Idioms

- An Idiom is a low-level patterns specific to a programming language.

Creational Design Patterns

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.



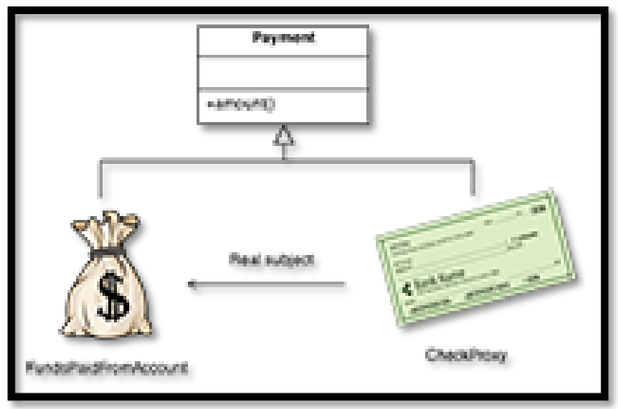
- **Abstract Factory**
  Creates an instance of several families of classes
- **Builder**
  Separates object construction from its representation
- **Factory Method**
  Creates an instance of several derived classes
- **Object Pool**
  Avoid expensive acquisition and release of resources by recycling 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

Structural Design Patterns

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

- **Adapter**
  Match interfaces of different classes
- **Bridge**
  Separates an object's interface from its implementation
- **Composite**
  A tree structure of simple and composite objects

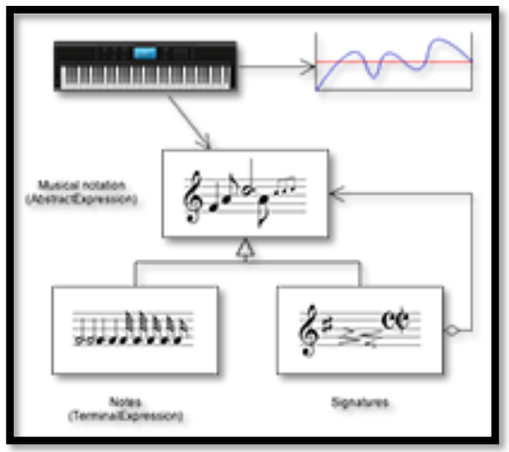
- **Decorator**
  Add responsibilities to objects dynamically
- **Facade**
  A single class that represents an entire subsystem
- **Flyweight**
  A fine-grained instance used for efficient sharing



Private Class Data
Restricts accessor/mutator access

- **Proxy**
  An object representing another object 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 chang

ITC311
Application Development and Emerging Technologies

UNIT 4. Prototyping and Quality Assurance

A. General Learning Outcome:

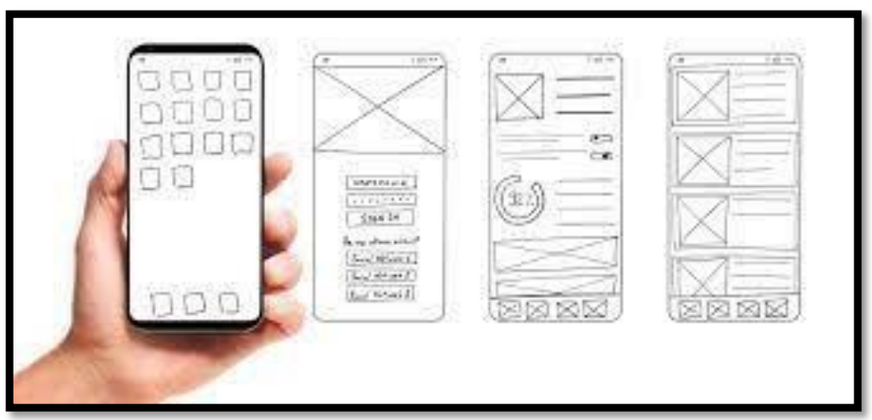
Discuss different computer hardware and software components. Specific Learning Outcomes:

1. Discuss the different prototyping processes
2. Implement quality assurance principles into real life software development situations
3. Differentiate ISO 9126 and ISO 25010.

B. Pre-Test

1. What is the importance of Prototyping process?
2. What are the different types of prototyping processes?
3. What are quality assurance principles?
4. Why is there ISO 9126 and ISO 25010?

C. Content



WHAT IS PROTOTYPING

It is a useful approach to requirement analysis and system development. It is highly considered by Management Information System (MIS) managers as a suitable solution to numerous executive problems and more useful in the development support of the applications.

It refers to an initial stage of a software release in which developmental evolution and product fixes may occur before a more significant release is initiated. These kinds of activities can

also sometimes be called a beta phase or beta testing, where an initial project gets evaluated by a smaller class of users before full development.

Prototyping is an essential part of the product development and manufacturing process required for assessing the form, fit, and functionality of a design before significant investment in tooling is made. It is used to allow the users to evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user-specific and may not have been considered by the developer during product design.

Prototyping aims to create harmony between the product requirements, ideas of designers, and users' mental models. It is achieved by using usability testing and evaluation methods and tools. If the interface does not fully meet the needs of the target audience and system requirements, the change of the prototype will be made much more comfortable than the final application (Stanovea, 2017).

PROTOTYPE

A 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. It is a draft version of a product that allows you to explore your ideas and show the purpose beyond a feature or the overall design concept to users before investing time and money into the development.

The prototype is developed and use to validate the requirements, can also be considered as guidance for the design phase and to provide idea about test case generation. The prototype is stored in a repository and reused in the future organization (Tanvir et al., 2017). It is an activity and a tool that has received considerable attention in the product development research communities in recent times (Elveruma, Welo & Tronvoll, 2016).

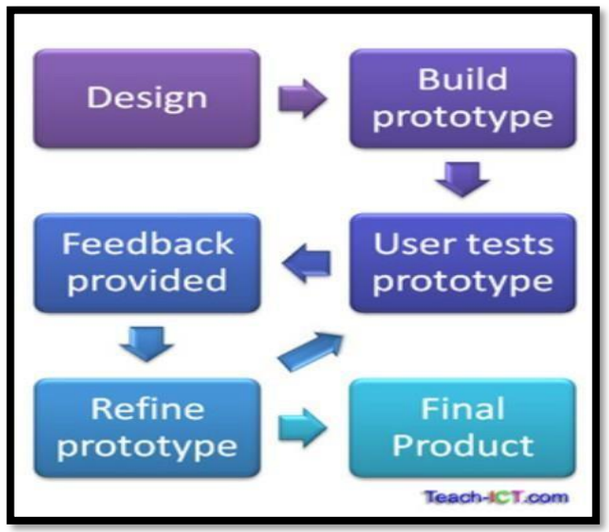


Figure 1. Prototype

A **prototype** denotes nearly aspect of the full system, let say, a mock-up or a scale or full-size model of a design or device of the graphical user interface. Consider a prototype GUI, where users click on command buttons and see its effect. The badge is not connected to a real system but is programmed by the developer to act as if it was. The command action is simulated with dummy data.

Now the user is beginning to see how the entire system will work before the developer spends even more time. So a prototype is NOT a fully working system, but it does provide an opportunity for the user to give feedback and suggestions for improvements. This may happen again and again until the full details are agree between developer and user. Therefore Prototyping is an 'iterative' process.

Benefits of Prototyping

- Better Quality System Delivered. Sometimes a developer may not fully understand what the end-user is expecting. Prototyping enables any misunderstandings to be identified and sorted out early on in the process.
- Identify Problems Early On. A working system is available early on in the process. The user can identify possible improvements that can be made before the system is completed.
- End-User Involvement. The end-user feels more involved in the development of the system and will 'buy' into it.
- Fulfill User Requirements. A system that has been through Prototyping will generally have an improved design quality and will be far closer to what the user needs.
- Cost Savings. It is far less expensive to rectify problems with the system in the early stages of development than towards the end of the project.
- Training. The prototype can eventually help train staff while the entire system is still being fully developed.

(https://www.teach-ict.com)

Best Practices of Prototype

Few things which you should know during the prototyping process:

- It would help if you used Prototyping when the requirements are unclear
- It is essential to perform planned and controlled Prototyping.
- Regular meetings are vital to keep the project on time and avoid costly delays.
- The users and the designers should be aware of the prototyping issues and
- At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.
- In the software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.
- You should select the appropriate step size for each version.
- Implement important features early on so that if you run out of the time, you still have a worthwhile system. (https://www.guru99.com)

TYPES OF PROTOTYPING MODEL

1. Rapid Throw away prototype

Throw away is based on the preliminary requirement. It is quickly developed to show how the condition will look visually. The customer's feedback helps drive changes to the situation, and the prototype is again created until the requirement is baselined.

Throw away Prototyping saves time and money for it is cheap, fast ones, which is designed to model an idea or feature. They are commonly used in the early phases of design when many ideas are still being considered. Throwaway prototypes may also be used in late-stage design in industries where products are launched at a low state of refinement.

Its objective is to ensure that the system requirements are validated and that they are clearly understood. As to Ganev, Dimitrova and Antonov (2018), rapid prototyping technology is a stylish manufacturing technology that creates material models from CAD data through 3D printers.

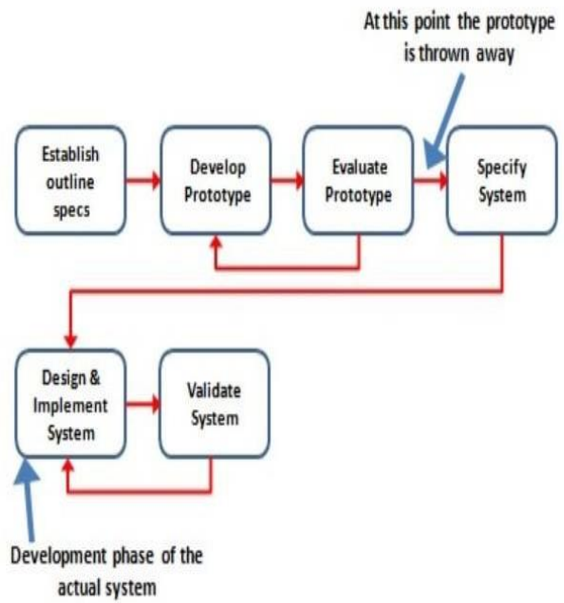
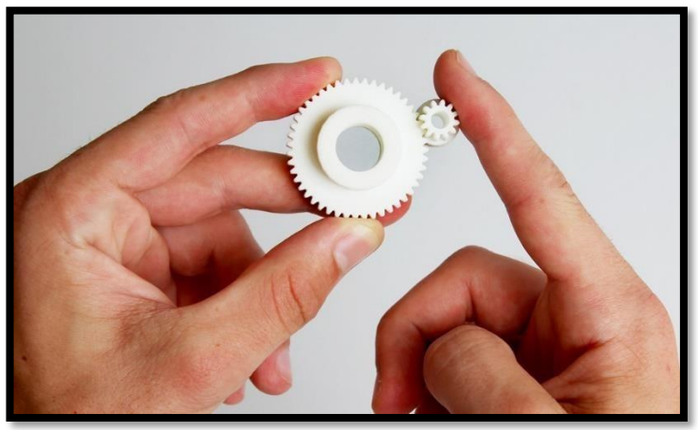


Figure 2. Throwaway Prototyping Model. (Khalid, 2018)

The throwaway prototype is NOT considered part of the final system. It is merely there to aid understanding and reduce the risk of poorly defined requirements. The full system is being developed alongside the prototypes and incorporates the changes needed.

The advantage of this approach is the speed with which the prototype is put together. It also focuses the user on only one aspect of the system, so keeping their feedback precise.

However, one difficulty of using throwaway Prototyping is that developers may be pressurized by the users to deliver it as a final system. The added issue is that all the working hours of putting together the Throwaway prototypes are lost, unlike the evolutionary approach. But the benefits may outweigh the disadvantages.



3D Prototyping

3D printing is considered an additive manufacturing method, meaning the material is added until a specific shape is formed. Depending on the level of accuracy required

3D printing can be beneficial for Prototyping in jewelry design, architecture, or engineering to make mechanical parts, cases, architectural models, props, and functional consumer products

- Creating product prototypes offers the opportunity to assess the functionality of an item and provide the chance to discuss room for improvement.
- And the art of creating useful prototypes is one that has only grown in terms of efficiency and quality with the rise of laser cutting.
- 3D laser cutting and 3D printers are revolutionizing the world of prototypes. It's mostly down to the fact that laser cutting is a precise art, and one with little room for error, and it's thanks to Computer-Aided Design (CAD) / Computer-

PROTOTYPE AND QUALITY ASSURANCE

- AD/CAM software works by instructing the laser cutter to perform precise movements. These movements are precise because CAD requires the user to put their exact measurements into the software, and then it can communicate your design with the laser cutting machine.
- This can be done quickly, and once the data is inputted, it doesn't need human supervision.
- The freedom and speed of cutting in this way mean prototype creation is suddenly a lot quicker than previously possible (Halliwell, 30 April 2020).

Evolutionary Prototype

Evolutionary Prototyping is a software development method where the developer or development team first constructs a prototype. After receiving initial feedback from the customer, subsequent prototypes are produced, each with additional functionality or improvements, until the final product emerges

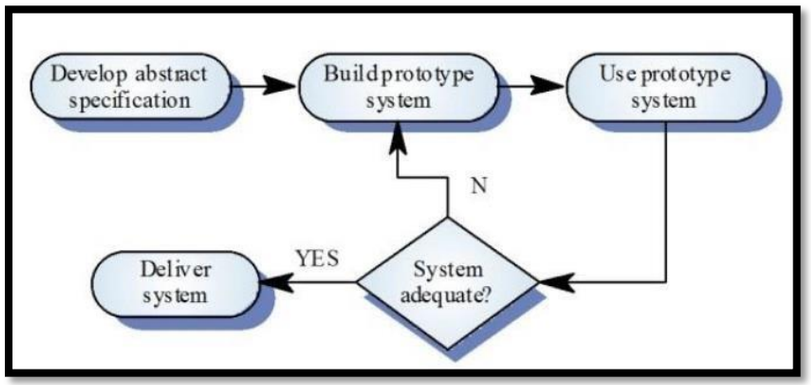


Figure 3. Evolutionary Prototyping

Also, the evolutionary prototyping scheme differs from the rapid or throwaway Prototyping, in that the developer begins with the best-understood requirements; whereas, in rapid Prototyping, the developer implements the least understood requirements. Furthermore, the prototype need not be built quickly.

Note that evolutionary Prototyping is similar to an incremental development in that parts of the system may be inspected or delivered to the customer throughout the software life cycle model (Sherrell, 2013).

PROTOTYPES CAN BE BOTH PHYSICAL AND VIRTUAL:

1. *Virtual prototyping* as a technology involves the creation and use of virtual models for presentation, testing and analyzing of particular details before creating the real physical models or optimizing parameters in parallel with physical prototypes.

2. *Physical prototyping* involves the creation of element models including functional ones in small quantities compared to the manufacturing ones. When a combination of the two technologies is possible significant reduction in costs and production time can be achieved with minimal risk of hidden deficiencies occurring in series production.

Physical prototyping involves the creation of a functional product model in small quantities compared to the final product quantitis in its industrialization and commercialization.

Phases of the Prototyping Model

Prototyping follows the SDLC (System Development Life Cycle) Phases.

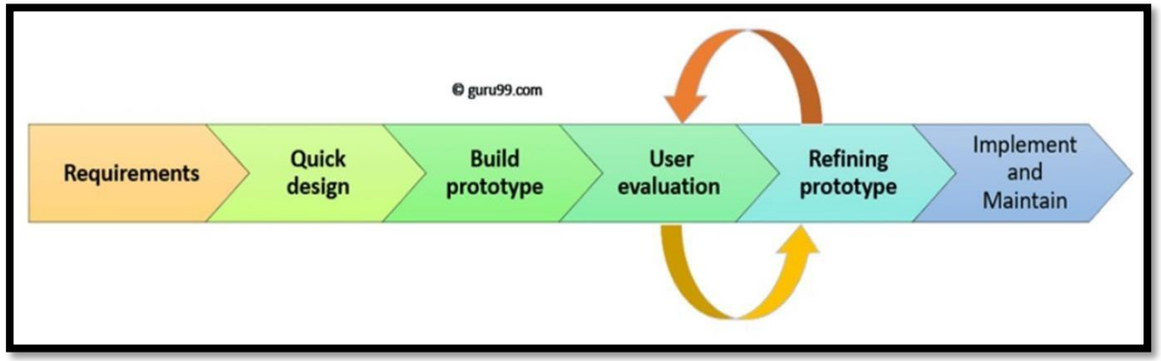


Figure 4. Prototyping Phases Model (guru99.com)

1. *Requirement Analysis*. A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. In this phase, the client is interviewed to determine his expected output in the system.

2. *Quick Design*. It is the initial design or short design. In this phase, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

3. *Build a Prototype*. In this phase, an actual prototype is designed based on the information gathered from a quick design. It is a small working model of the required system.

4. *User Evaluation*. In this phase, the proposed system is presented to the client for an initial evaluation. At this level, it identifies the strength and weaknesses of the working model. Comment and suggestions are collected from the customer and provided to the developer.

5. *Refining Prototype*. If the client is not satisfied with the new prototype, refining is needed according to the user's feedback and suggestions. When the client conformed to the developed prototype, a final system is developed based on the approved final prototype.

6. *Implement the Product and Maintain*. When the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

WHAT IS QUALITY?

Quality is tough to define, and it is stated: "Fit for use or purpose." It is all about gathering the needs and expectations of customers concerning functionality, design, reliability, durability, & price of the product.

Software quality refers to the degree a system, component, or process conforms to specific requirements or expectations. In many cases, end-users know "quality" software when they see it. This software is easy to use and error-free. It also enables users to perform tasks quickly and effortlessly, on any device, and at any time (Mcconville, April 2018).

WHAT IS ASSURANCE?

Assurance is nothing but a constructive declaration on a product or service, which gives confidence. It is the certainty of a product or a service that will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

WHAT IS QUALITY ASSURANCE (QA)?

Quality Assurance is a procedure to safeguard the quality of software products or services provided to the clients by an organization. Quality assurance emphasizes on refining the software development process and making it efficient and effective as per the quality standards defined for software products. Quality Assurance is well known as QA Testing.

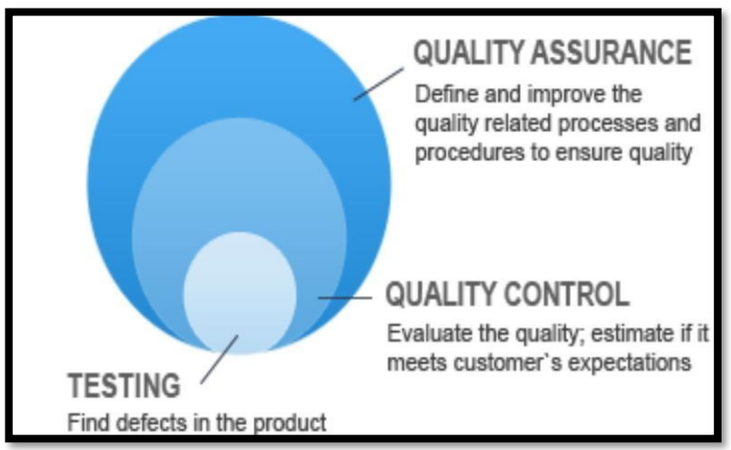


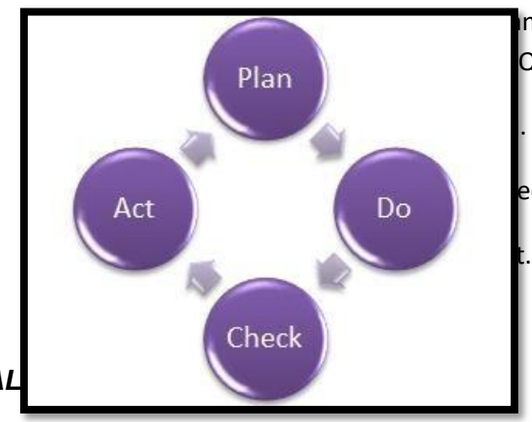
Figure 5. Quality Assurance

Quality Assurance checks whether the product developed is fit for use. For that, the organization should have processes and standards to be followed, which need to be improved periodically. It concentrates primarily on the quality of the product/service that we provide to the customers during or after the implementation of the software.

PROCESS OF QUALITY ASSURANCE (QA)?

Quality Assurance (QA) starts and sustains set requirements for developing or manufacturing reliable products. A quality assurance system is meant to increase customer confidence and a company's credibility, while also improving work processes and efficiency, and it enables a company to better compete with others.

Quality assurance has a defined cycle called the PDCA cycle or the Deming cycle. The phases of this cycle are:



n. The organization must plan and establish the process related objectives and determine the operations required to deliver a Quality end product.

. Development and testing of Processes and also "do" changes in the processes

eck. - Monitoring of processes, modifythe processes and check whether it meets the predetermined objectives

t. Implement actions that are necessary to achieve improvements in the processes

QUAL

There are five crucial Quality Assurance Functions:

1. *Technology transfer*. It involves getting a product design document as well as trial and error data and its evaluation. The documents are distributed, checked, and approved.

2. *Validation*. It validates the principal plan for the entire system to be prepared. Approval of test criteria for validating product and process is set. Resource planning for the execution of a validation planis done.

3. *Documentation*. Documentation controls the distribution and archiving of documents. Any change in a record is made by adopting the proper change control procedure—approval of all forms.

4. Assuring quality of products

5. Quality improvement plans

QUALITY ASSURANCE CERTIFICATIONS

There are numerous certifications available in the industry to ensure that Organizations follow Standards Quality Processes. Customers make this as qualifying criteria while selecting a software vendor.

ISO 9000

The ISO (International Organization for Standardization) was first established in 1987, and it is related to Quality Management Systems. This standard helps the organization ensure quality to its customers and other stakeholders. An organization who wishes to be certified as ISO 9000 is audited based on their functions, products, services, and their processes. The main objective is to review and verify whether the organization follows the procedure as expected and check whether existing processes need improvemen

Requirements Analysis and Modeling

Prepared by:

CHRISTIAN I. CABRERA
College of Computer Studies
E-mail Address: krizchan31@gmail.com

College of Computer Studies

Learning Objectives

Click to edit

Specific Learning Outcomes:

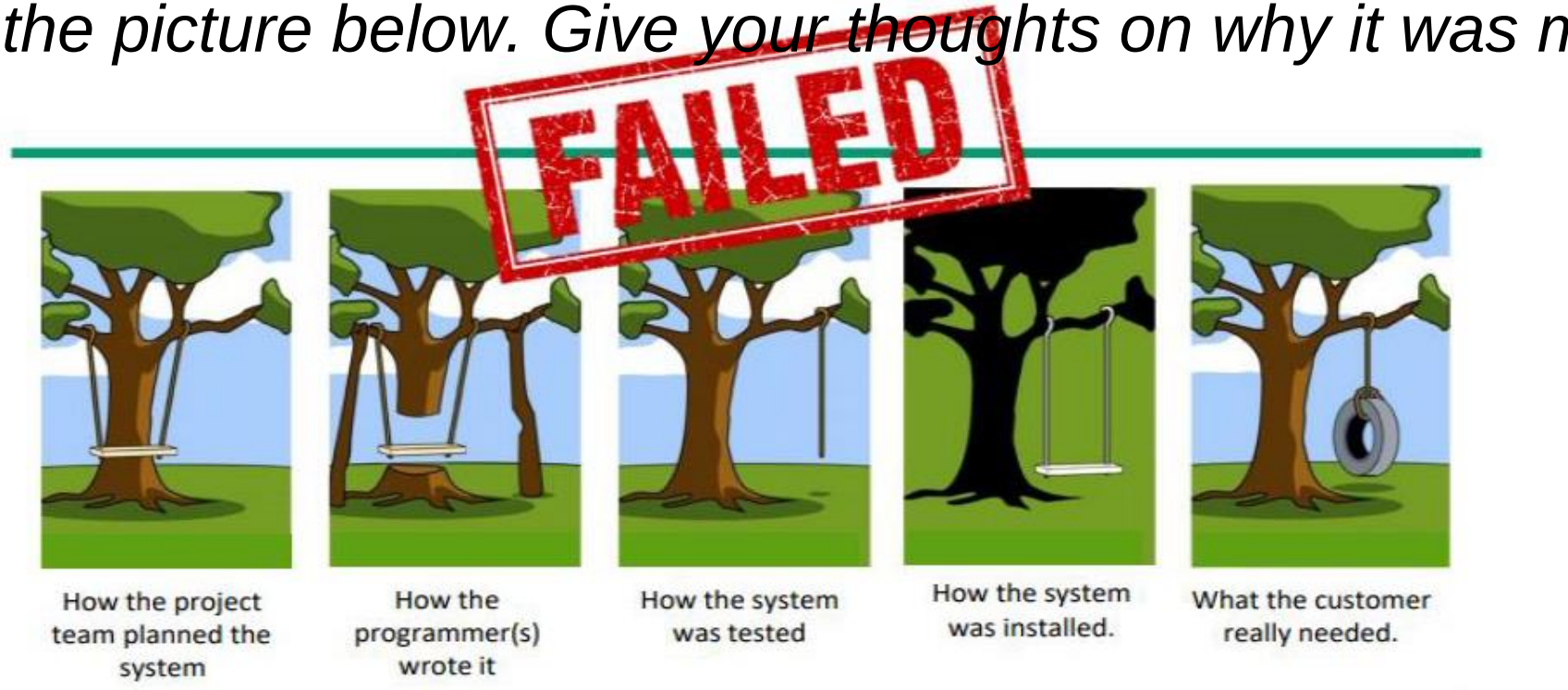
- Create a detailed requirements definition report that lists functional and non-functional requirements.
- Collect information of application requirements using the requirements-gathering techniques.
- Create functional models of business processes/requirements using use-case diagrams.

College of Computer Studies

Click to edit Requirements Analysis and Modeling

Pre-Test

Consider the picture below. Give your thoughts on why it was marked FAILED?



College of Computer Studies

Stages in Software Development

Click to edit

System development or application development is a systematic process of defining, designing, testing, and implementing a new application. The four (4) fundamental phases typical to all software development projects are **planning, analysis, design, and implementation**.

College of Computer Studies

Click to edit

Learning Objectives

In the analysis phase, all projects require a system analyst or requirements engineer. The role of the systems analyst is to gather requirements, analyze the gathered requirements, model the user needs, and create blueprints for how the system should be built. The first activity of a system analyst is to determine the user requirements for a new system and refine them into detailed requirements.

College of Computer Studies

Requirements Analysis and Modeling

Click to edit

Requirements Determination

- The purpose of **requirements determination** is to convert the high-level requirements defined from user requirements into a list of detailed requirements. It is called **requirements definition report**, that can be used as inputs for creating models such as functional model.

A **requirement** is simply a statement of what the system must perform or what characteristic it must have.

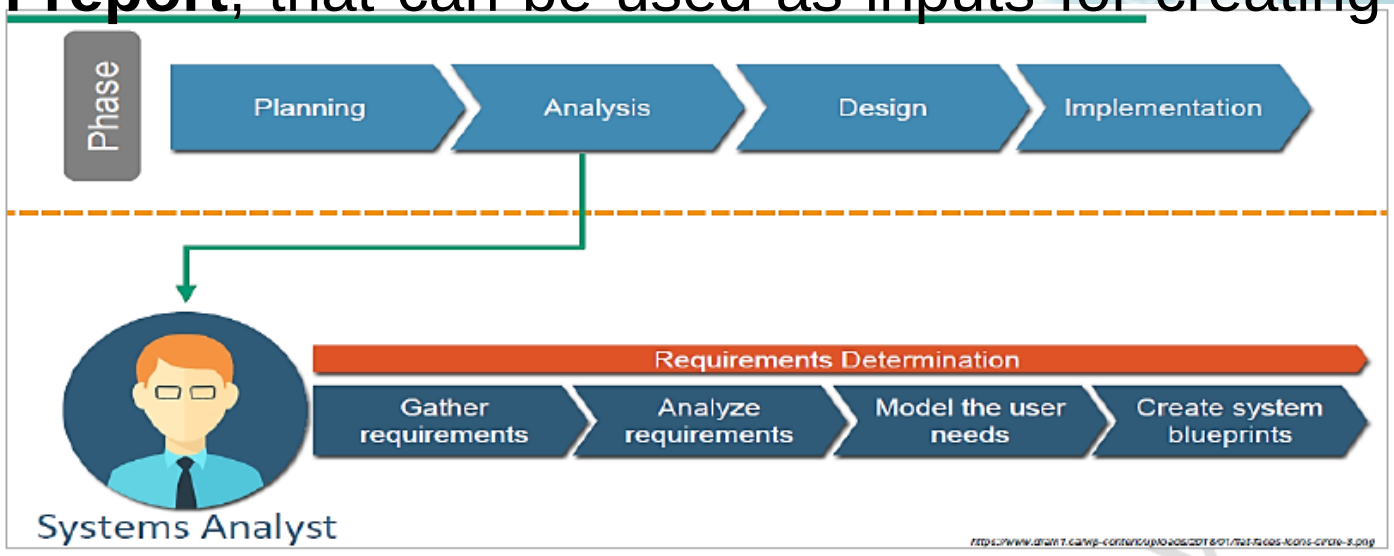


Figure 1. Stages of Software Development

College of Computer Studies

Requirements Analysis and Modeling

Click to edit

User requirements are determined from discussions with the client and determine their actual needs. Then, these are refined in the design phase into system requirements that state the functional and non-functional requirements of the system.

During analysis, requirements are written from the perspective of the needs of the user. In the design phase, the user requirements will describe how the system will be implemented.

Requirements can be categorized as **functional and non-functional**.

College of Computer Studies

Categories of Requirements

Click to edit Master title

Functional Requirements- These are requirements directly related to the process a system has to perform or the data it needs to contain. For example, requirements stating that a system must have the ability to search for available products, report actual and budgeted expenses, or generate financial statements are all functional requirements.

College of Computer Studies

Categories of Requirements

Click to edit Master title

Let say you want to develop a tourist guide app for Oriental Mindoro. The following are the sample functional requirements:

- The application should be able to provide a detailed map of every towns of Oriental Mindoro with proper labels.
- The application should be able to provide 3D views of some tourist spots in Oriental Mindoro.
- The user should be able to easily the destitation they want.

THINK ABOUT!

What are the possible functional requirement of the Tourist Guide App for Oriental Mindoro?x

College of Computer Studies

Categories of Requirements

Click to edit Master title

Non-functional requirements- These are requirements that pertain to behavior properties that a system must have, including operational, performance, security, and cultural and political. These requirements define how a system or software is supposed to be or its system properties.

College of of Computert Studies

Categories of Requirements

Click to edit Master title

Non-functional requirements

- **Performance requirements-** these deal with issues related to performance such as response time, capacity, and reliability of the system.

  For example, The map should load in less than 3 seconds.

- **Operational requirements-** specify the operating environment(s) in which the system must perform and how these might change over time. These usually refer to operating systems, system software, and information systems with which the system must interact. These might also include the physical environment, such as in what building area the system must be installed.

For example, The app should be compatible with any mobile operating system.

College of of Computert Studies

Categories of Requirements

Click to edit Master title

Non-functional requirements

- **Security requirements-** these address issues with security, such as who has access to the system's data and must have the ability to protect data from disruption or data loss.

For example, the user's information must be encrypted.

- **Cultural and political requirements** deal with issues related to the cultural and political factors and legal requirements that affect the system.

For example, the app should use English and Tagalog language only.

College of Computer Studies

Click to edit

Requirements Definition Report

The **requirements definition report,** also referred to as **requirements specification document or software requirement specification (SRS)**, lists the functional and non-functional requirements in an online format and defines the scope of the system or software. This is used to describe what the system can do, including its characteristics, and is used which serve as a guide in developing the system or software. Below is a sample of the requirements definition report.

SRS

- Revision History

1. Introduction
   - 1.1. Document Purpose
   - 1.2. Product Scope
   - 1.3. Product Overview
   - 1.4. Definitions
2. References
3. Specific Requirements
   - 3. 1. Functional Requirements
   - 3. 2. Non-functional Requirements
     - 3.2.1. Operational Requirements
     - 3.2.2. Performance Requirements
     - 3.2.3. Security Requirements
     - 3.2.4. Cultural Requirements
4. Verification
5. Appendixes

Figure 2. Sample SRS Template

A properly written requirements definition report describes exactly what the system needs to the analyst and benefits all successive phases of the software development process, such as software testing. When discrepancies arise during software or system development, this document serves as a tool for clarification.

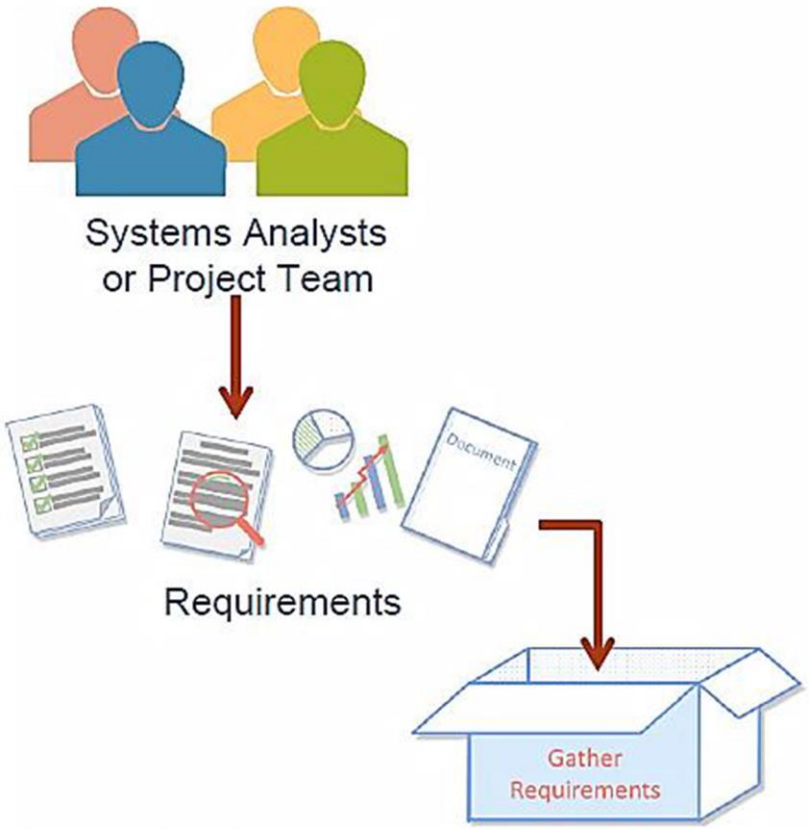
The SRS format varies depends on the organization/ company using it. Some are using an International Organization for Standardization (ISO) standards and Institute of Electrical and Electronics Engineers (IEEE) standards or create their own based on their best practices.

Information Technology 14

College of Computer Studies

Click to edit

Requirements Gathering Techniques



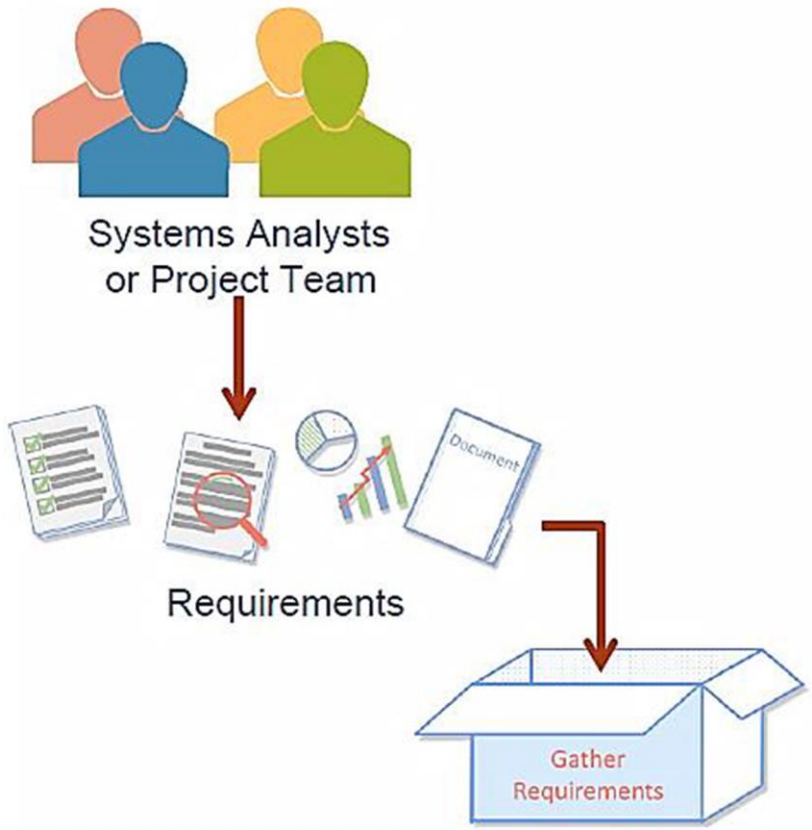
Requirements gathering or requirements elicitation is the process of cooperating with clients or users to determine what requirements are needed.

When determining requirements for the requirements definition report, the system analysts use requirements-gathering techniques to collect information and list the business or user requirements defined from that information. The analysts then work with the entire project team and clients to verify, change, and complete the identified requirements before moving to design.

College of Computer Studies

Click to edit

Requirements Gathering Techniques



1. Interviews
2. Joint Application Development (JAD)
3. Questionnaires
4. Document Analysis
5. Observation

College of Computer Studies

click to edit

Requirements Gathering Techniques

There are many techniques for gathering requirements. Each technique has its strengths and weaknesses, many of which are complementary, so most projects use a combination of techniques. The five (5) commonly used techniques are the following:

1. Interviews- This is the most commonly used requirements-gathering technique. In general, interviews comprise interviewers and interviewees.

Figure 3. Interview Process

College of Computer Studies

Interviews

Click to edit Master title style

There are five (5) basic steps to the interview process, which are as follows:

Step 1: Selecting Interviewees- The first step in interviewing is creating an interview schedule that lists who will be interviewed, when, and what purpose.

The people who appear on the interview schedule are selected based on the analyst's information needs. The client, users of the system, and other project team members can help the analyst determine who in the organization can best provide valuable information about the requirements

Name	Position	Purpose of Interview	Meeting
Rachel Simmons	Director, Accounting	Strategic vision for the new accounting system	Monday, March 1 8:00 – 10:00 AM
Linda Aquino	Director, Human Resources	Understand reports produced for Human Resources by the current system; determine information requirements for future system	Monday, March 1 2:00 – 3:30 PM
Jennifer Dela Cruz	Manager, Accounts Receivable	Current problems with accounts receivable process; future goals	Monday, March 1 4:00 – 530: AM
Mark Torres	Manager, Accounts Payable	Current problems with accounts payable process; future goals	Wednesday, March 3 10:30 – 11:30 AM
Anne Castro	Supervisor, Data Entry	Accounts receivable and payable processes	Wednesday, March 3 1:00 – 3:00 PM

Take Note!

People at different levels of the organization have varying perspectives on the system. Therefore, it is important to include managers who manage the process and the employee who actually performs to gain high-level perspective on an issue.

Figure 3. Sample of An Interview Schedule

College of Computer Studies

Interviews

Click to edit Master title style

Step 2. Designing Interview Questions- There are three (3) types of interview questions:

- **Close-ended questions** are those that require a specific answer. An analyst may ask, "How many requests does a company process per day?'
- **Open-ended questions** require the interviewee to answer in open text format where they can answer based on their complete understanding and knowledge. These interview questions are designed to gather rich information and give the interviewee more control over the information revealed during the interview. For example, the analyst could ask the interviewee, "What are some improvements you would like to see in the new system?"
- **Probing questions** follow up on what has just been discussed from close-ended or open-ended questions to learn more. These are often used when the interviewer is unclear about an interviewee's answer. These interview questions encourage the interviewee to expand on or confirm information from a previous response.

Take Note!

These types of interview questions are usually combined during an interview. Whatever type of interview is conducted, interview questions must be properly organized and listed into a logical sequence in a formal list so that the interview flows well.

For example, after the analyst asked them, "What are some improvements you would like to see in the new system?" the analyst will ask a probing question, "Why do you think those improvements are necessary for the new system?"

College of Computer Studies

Interviews

Click to edit Master title style

Step 3. Prepare for the Interview – An interviewer needs to prepare for the interview. The interviewer should have a general interview plan listing the questions in the appropriate order and anticipate possible answers and provide follow-up with them.

Step 4. Conduct the Interview – The interviewer should explain why s/he is conducting the interview and explain why the interviewee is there. The interviewer may proceed with the planned interview questions and should take notes to record the interview. Always ask permission if it is appropriate for him/her to record the interview. After conducting the interview, the interviewer should briefly explain what will happen to the system.

Step 5. Post-Interview Follow-up – After the interview, the analyst must prepare an interview report that describes the information from the interview. The report must contain interview notes, information that was collected throughout the interview, and a summary in a helpful format. The interview report must be sent to the interviewee with a request to read it and inform the analyst for clarifications or updates.

College of Computer Studies

Interviews

Click to edit Master title style

Person Interviewed: Linda Aquino, Director, Human Resources

Interviewer: Katherine Castro

Purpose of Interview:

- Understand reports produced for Human Resources by the current system; and
- Determine information requirements for future system.

Summary of Interview:

- Sample reports of all current HR reports are attached to this report. The information that is not used and missing information are noted on the reports.
- Two (2) biggest problems with the current system are as follows:
  1. The data are too old (the HR Department needs information within two (2) days of month end; currently, information is provided to them after a three-week delay); and
  2. The data are of poor quality (often reports must be reconciled with the departmental HR database).
- The most common data errors found in the current system include incorrect job level information and missing salary information.

Open Items:

- Get current employee roster report from May Santos (telephone extension 435).
- Verify calculations used to determine vacation time with May Santos.
- Schedule interview with Jimmy Lee (telephone extension 237) regarding the reasons for data quality problems

Figure 4. Sample Interview Report

Source: Systems analysis & design. An object-oriented approach with UML (5th ed.), 2015. p. 101

College of Computer Studies

Joint Application Development (JAD)

1. **Joint Application Development (JAD)** –This is the most useful method for collecting information from clients and is a structured process that involves a group of participants to meet together under the direction of a facilitator.

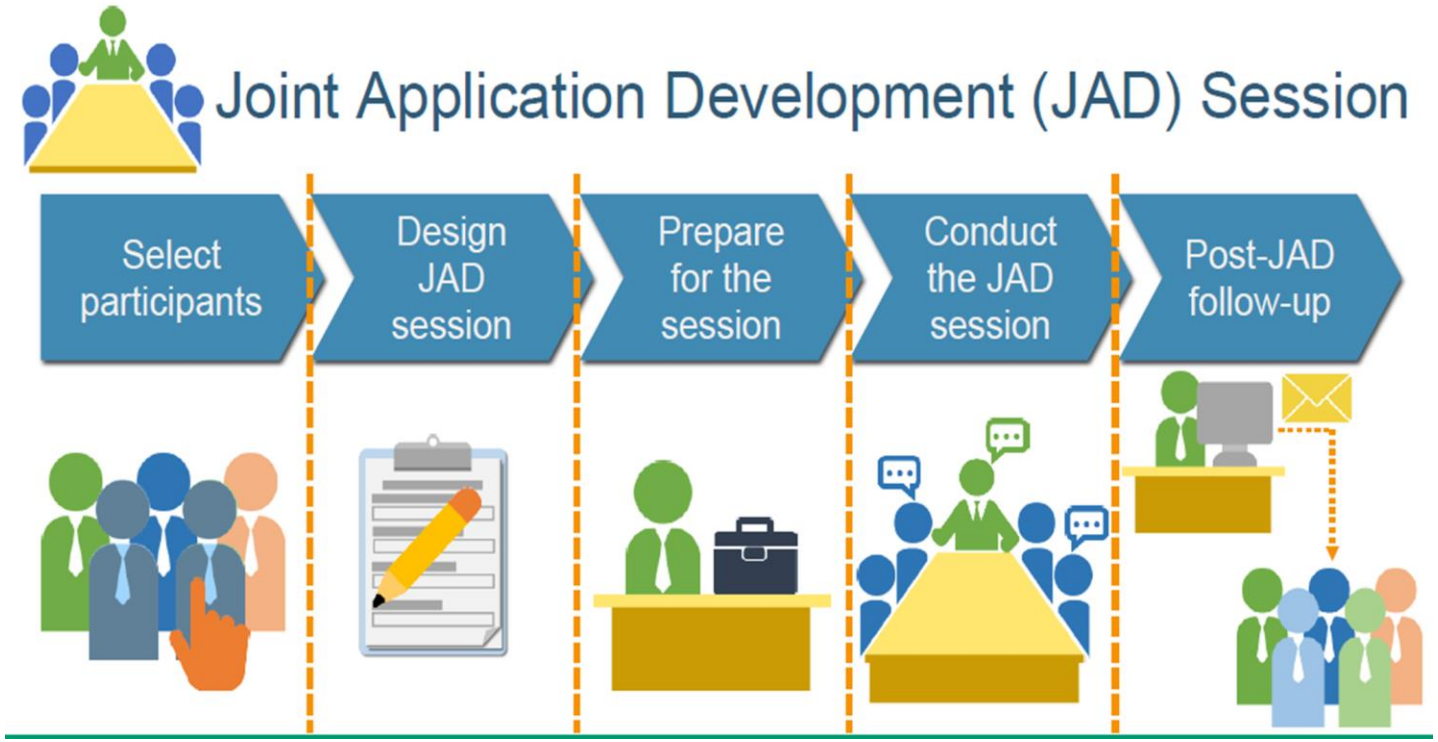
   There must be someone called "scribes" to assist the facilitator by recording notes, making copies, and so on. **Scribes** can use computers and CASE tools to record information as the JAD session proceedings.

College of Computer Studies

click to edit

Joint Application Development (JAD)

There are five (5) basic steps to the JAD approach:



Step 1. Selecting Participants – The process of selecting participants for a JAD session is the same as selecting an interviewee in the interview technique. Selecting participants must be based on the information they can contribute to provide a broad mixture of organization levels and build political support for the new system.

Step 2. Designing JAD Session – The timing cover of the JAD session depends on the size and scope of the project the session can. For example, the participants and analysts can create analysis deliverables collectively, such as the functional models or requirements definition. These are also designed to

collect specific information from the participants/users and require developing questions before the meeting.

College of Computer Studies

click to edit

Joint Application Development (JAD)

Step 3. Preparing for the JAD Session – JAD sessions can go beyond the depth of a typical interview, so it is essential to prepare the analysts and participants for a JAD session. The participants must understand what is expected of them. For example, suppose the goal of the JAD session is to develop an understanding of the client's current system. In that case, the participants must bring procedure manuals and documents with them.

Step 4. Conducting the JAD Session – JAD sessions follow a formal agenda and rules that define appropriate behavior. The standard rules include following the schedule, respecting other's opinions, accepting disagreement, and ensuring that only one (1) person talks at a time. During the session, the JAD facilitator's scribes must record notes, make copies, etc.

Step 5. Post-JAD Follow-up – JAD post-session report is prepared and circulated among the participants. Because JAD sessions are longer and provide more information, it usually takes a week or more after the JAD session to create a report.

College of Computer Studies

Questionnaires

Click to edit Master title style

Questionnaires –These are often used when there are many people from whom information and opinions are needed. A questionnaire is a set of written questions used to obtain information from individuals. Administering questionnaires are a common technique with systems or software intended for use outside the organization, such as customers or vendors. These can be distributed in paper form or electronic form, either through e-mail or on the Web.

College of Computer Studies

Questionnaires

Click to edit Master title style

There are four (4) steps when using questionnaires as a gathering technique:

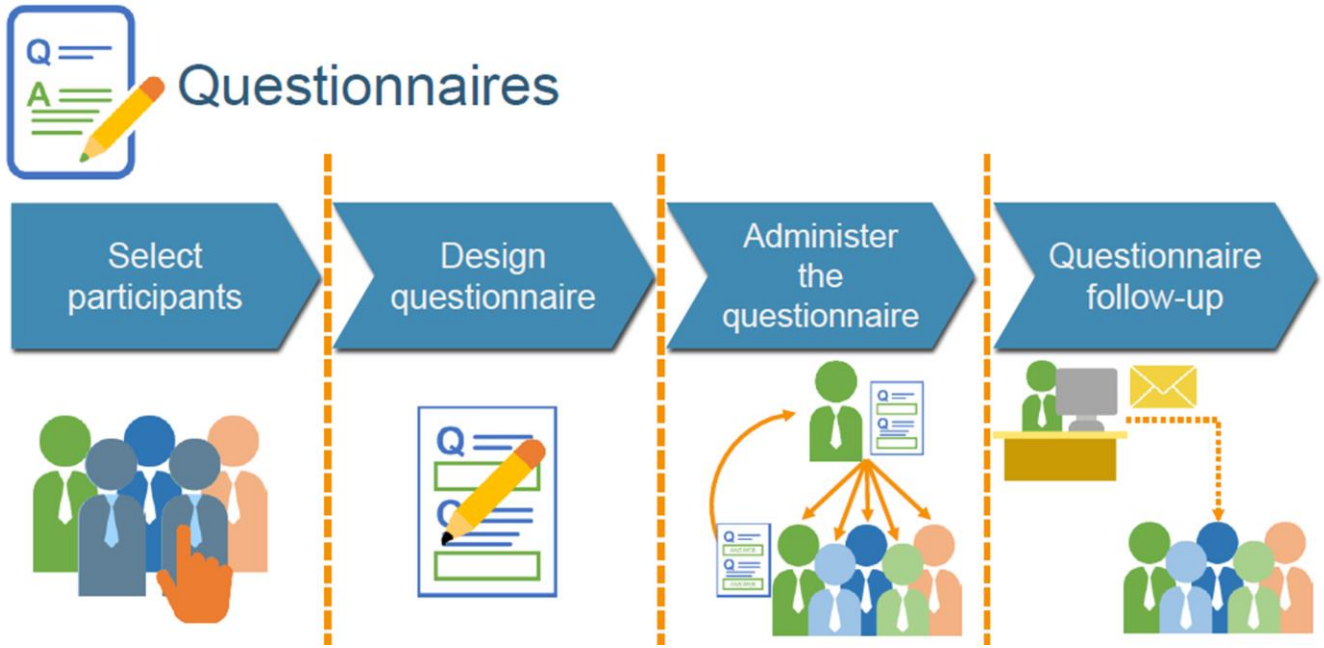


Figure 6: Gathering data through questionnaires

Step 1. Selecting Participants– As with interviews and JAD sessions, the first step is to identify the individuals to whom the questionnaires will be sent. When selecting participants, the standard approach is to select a sample (a subset of the population chosen to represent the population under study) of people who represents an entire group.

Step 2. Designing a Questionnaire – Questions on questionnaires must be clearly written and leave little room for misunderstanding, so closed-ended questions are mostly used. Questions must enable the analyst to separate facts from opinions. Before distributing the questionnaires, the analyst must be clear of how the collected information will be analyzed and used. Questions should be relatively consistent in style. Thus, it is generally a good practice to group related questions to make them

simpler to answer.

College of Computer Studies

Questionnaires

Click to edit Master title style

Step 3. Administering the Questionnaire – When administering the questionnaires, the key issues are how to make sure that participants will complete the questionnaire and send it back. To improve the response rate, explain to the participants why the questionnaire is being conducted and why the respondent has been selected, stating a date by which the questionnaire is to be returned. Some analysts personally hand out the questionnaire and contact those who have not returned the questionnaire after the deadline.

Step 4. Questionnaire Follow-up – It is essential to process the returned questionnaires and develop a report soon after the questionnaire's deadline. This ensures that the analysis process proceeds in a timely fashion and that respondents who requested copies of the results receive them promptly.

College of Computer Studies

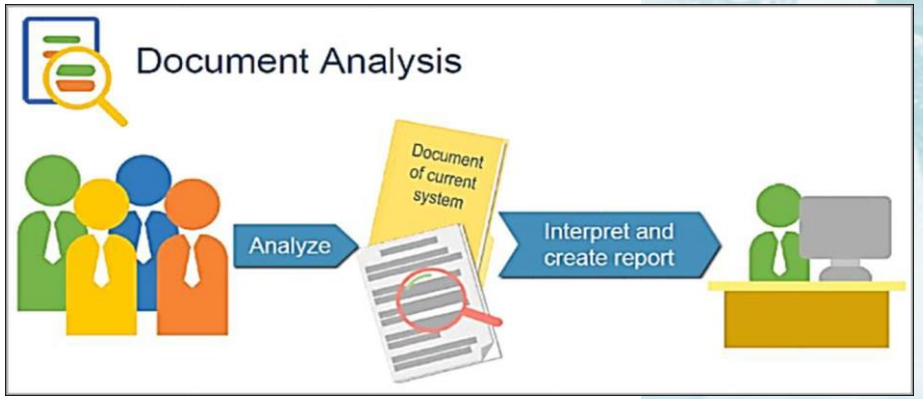
click to edit

Document Analysis

Document Analysis – This is often used by project teams to understand the current system (as-is system) of an organization. This is a form of qualitative research in which the researchers analyze and interpret documents to give voice and meaning to an assessment topic.

Many helpful documents exist in an organization that requirements analysts can review, such as paper reports, memorandums, policy manuals, user-training

Document analysis is done in which documents, both printed and electronic, are reviewed to gather information.



manuals, organization charts, forms such as registration forms, receipts, the user interface of the current system, and even their website.

Figure 7: Document analysis process

College of Computer Studies

Observation

Click to edit Master title style

Observation –This is a powerful tool for gathering information about the current system (as-is system) because it enables the analyst to see the reality of a situation rather than listening to others describe it in interviews or JAD sessions. In most cases, observation supports the information that users provide in interviews

Figure 8. 8: Observation process

or JAD sessions. When it does not, it is a meaningful sign that extra care must be taken in analyzing the business system or software.

College of Computer Studies

Selecting the Appropriate Data Gathering Technique

Click to edit Master title style

In practice, most projects use a combination of techniques. These are the following characteristics that describe the strengths and weaknesses of each technique:

- **Type of information** – Some techniques are more suited for use at different stages of the analysis process, namely: understanding the three (3) stages: current system (as-is system), identifying improvements, or developing a new system (to-be system).
  - Interviews and JAD sessions are commonly used for the three (3) stages above
  - The document analysis and observation are usually most helpful for understanding the as-is system – although occasionally, they provide information about current problems that need to be improved.
  - Questionnaires are often used to gather information about the as-is system as well as general information about improvements.
- **Depth of Information** refers to how rich and detailed the information is that the technique usually produces and the extent to which the technique helps obtain facts and opinions and understand why those facts and opinions exist.

- Interviews and JAD sessions are beneficial for providing a good depth of rich and detailed information and helping the analyst understand the reasons behind them.
- Documents analysis and observation help obtain facts but little beyond that.

Selecting the Appropriate Data Gathering Technique

- **Breadth of Information** refers to the range of information and information sources that can be easily collected using the chosen technique.
  - Questionnaires and document analysis are both easily capable of soliciting a wide range of information from a large number of information sources.
  - Interviews and observation require analysts to visit information sources individually and therefore take more time.
  - JAD sessions are in the middle because many information sources are brought together at the same time.
- **Integration of Information** –Combining the gathered information from different sources and attempting to resolve differences in opinions or facts are usually very time consuming because it means contacting each information source in turn, explaining the discrepancy, and attempting to refine the information when in fact it is another user in the organization who is doing so.
  - All techniques suffer integration problems to some degree, but JAD sessions are designed to improve integration because all information is integrated when it is collected during the session. The immediate integration of

information is one of the most important benefits of JAD that distinguishes it from other techniques, and this is why most organizations use JAD for important projects.

College of Computer Studies

Selecting the Appropriate Data Gathering Technique

Click to edit Master title style

- **Breadth of Information** refers to the range of information and information sources that can be easily collected using the chosen technique.
  - Questionnaires and document analysis are both easily capable of soliciting a wide range of information from a large number of information sources.
  - Interviews and observation require analysts to visit information sources individually and therefore take more time.
  - JAD sessions are in the middle because many information sources are brought together at the same time.
- **Integration of Information** –Combining the gathered information from different sources and attempting to resolve differences in opinions or facts are usually very time consuming because it means contacting each information source in turn, explaining the discrepancy, and attempting to refine the information when in fact it is another user in the organization who is doing so.
  - All techniques suffer integration problems to some degree, but JAD sessions are designed to improve integration because all information is integrated when it is collected during the session. The immediate integration of

information is one of the most important benefits of JAD that distinguishes it from other techniques, and this is why most organizations use JAD for important projects.

College of Computer Studies

Selecting the Appropriate Data Gathering Technique

Click to edit Master title style

- **User Involvement** – This refers to the amount of time and energy the intended users of the new system must devote to the analysis process. However, user involvement can have a high cost, and not all users are willing to contribute valuable time and energy.
  - Questionnaires, document analysis, and observation place the least burden on users.
- **Cost –** This defines the cost to spend when using each technique and does not imply if a technique is more or less effective than the other techniques.
  - Questionnaires, document analysis, and observation are low-cost techniques, although observation can be pretty time-consuming. Interviews and JAD sessions require the effort of the users.

College of Computer Studies

Selecting the Appropriate Data Gathering Techniques

Click to edit Master title style

Consideration	Interviews	Joint Application Development	Questionnaires	Document Analysis	Observation
Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low
Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low to Medium	Low	Low	Low to Medium

Source: Systems analysis & design. An object-oriented approach with UML (5th ed.), 2015. p. 108

College of Computer Studies

Click to edit Master title Requirement Analysis

After gathering data, the **requirements analysis** is conducted to determine if the specified requirements are actually required. This is the process by which gathered requirements are analyzed to ensure that those requirements are required to solve conflicts that will affect the designing and development of the system or software.

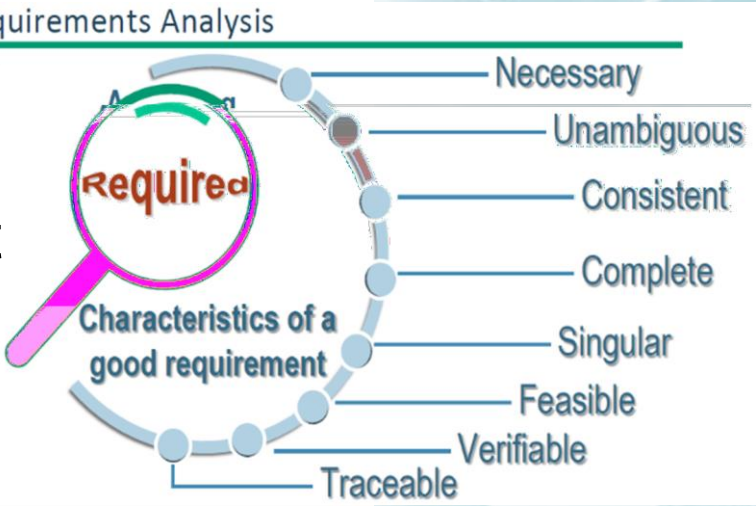


Figure 9. Characteristics of Good requirements

Requirements should be analyzed for the characteristics of reasonable requirements and project teams, or analysts employ a checklist as a tool to determine if each specified requirement is required. Requirements analysis must be performed by a project manager, systems analysts,

TAKE NOTE!

Requirements are the foundation of the system, and project planning and development is based on the elicited requirements. Therefore, it is essential that each elicited requirement has the characteristics of a good requirement.

developers, quality assurance analysts or software testers, and other relevant team members.

College of Computer Studies

Characteristics of Good Requirements

Click to edit Master title style

- **Necessary –** The requirement is needed or specifies an essential factor on the system. This characteristic defines that if the requirement is removed, a deficiency will exist; other features of the system cannot achieve that.
- **Unambiguous –** The requirement is stated that it should only be interpreted in only one (1) way. This characteristic defines that requirement is specific and easy to understand.
- **Consistent –** The requirement should not have conflicts with other requirements. Any conflicts must be determined, and overlap between requirements must be resolved.
- **Complete –** The requirement must completely describe the necessary functionality to meet the user's need.
- **Singular –** The requirement statement includes only one (1) requirement with no use of conjunctions. Requirement statements with conjunctions such as "and," "or," and "but" must be reviewed carefully to

see if they can be broken into singular requirements. The requirement must not contain multiple needs to avoid anomalies and for an analyst to able to analyze it independently.

College of Computer Studies

Characteristics of Good Requirements

Click to edit Master title style

- **Feasible –** The requirement is achievable within system constraints such as time, cost, legal, and available resources. This characteristic defines that the requirement is possible to be implemented.
- **Verifiable –** The requirement is verifiable if the implemented system or software can be tested to prove that the specified requirement has been met. Different testing tools are used to prove that the system or software satisfies the specified requirement. Verifiability is enhanced if the requirement is measurable by analysis, inspection, and test cases.
- **Traceable –** A requirement is traceable if it satisfies all the other characteristics of a good requirement. The requirement must have a unique identifier to trace all changes to it throughout

the development life cycle. An example of a unique identifier for a requirement can be "REQ001."

College of Computer Studies

Characteristics of Good Requirements

Click to edit Master title style

Table 2. Characteristics of good requirements with examples of bad and good requirements.

tion of Good
ents

Assuming the system is online student portal mobile application: REQ001 – The student shall log in to the system by providing his/her student number and password and by scanning his/her fingerprint. This requirement may not be necessary since some smartphones and computers have no built-in fingerprint sensor, or the client verified that it is not necessary for their process. This requirement must be verified and resolved before designing the system or software.	REQ001 – The student shall log in to the system by providing his/her student number and password. REQ002 – If the student's device is equipped with a fingerprint sensor, s/he shall log in to the system by scanning his/her fingerprint.
REQ001 - The system shall automatically refresh the databases quickly. This requirement is ambiguous since the word "quickly" is ambiguous. The specific time must specify to avoid ambiguity in the statements.	REQ001 - The system shall automatically refresh the databases within 0.5 seconds.
REQ001 – The system shall accept PayPal payment. REQ002 – The system shall accept only credit card payments. These requirements show conflicts since REQ001 overlaps REQ002. Overlaps between these requirements must be resolved.	REQ001 – The system shall accept PayPal payment. REQ002 – The system shall accept credit card payments.

College of Computer Studies

Characteristics of Good Requirements

Click to edit Master title style

REQ001 – The registrar shall be able to enroll a student in an undergraduate course and drop that enrolled undergraduate course.
This statement contains two (2) requirements that must be broken down into singular requirements.

REQ001 – The registrar shall be able to enroll a student in an undergraduate course.

REQ002 – The registrar shall be able to drop a student from their enrolled undergraduate course.

REQ001 – The employees shall log in to the system by examining their deoxyribonucleic acid (DNA).
This requirement may not be feasible within the available resources and budget of the company or clients.

REQ001 – The employees shall log in to the system by scanning their fingerprints.

REQ001 – The user interface of the system shall be user-friendly.
This requirement statement is not testable or measurable since it's ambiguous and not complete.
This statement needs to be resolved and must be verified by the clients. This can be fixed by adding details of how it will be tested.

REQ001 – The system shall display the current total enrolled students on the dashboard page every two (2) seconds.

REQ002 – The user interface of the system shall use the Arial font

College of Computer Studies

Click to edit Master title style Characteristics of Good Requirements

"These are the standard characteristics that a requirement must have, but project teams can add other characteristics depending on how they will satisfy the actual requirements needed by client. For example, a project team can add a characteristic that the requirement must not conflict with the policy of the company or does the requirement align with the business goals of the company.

After analyzing the requirements, the analyzed requirements must be systematically organized to create the requirements definition report and must be reviewed by the client for approval. Creating requirements definition is an iterative process where the

analysts or project team elicit information, analyze the information, then create or revise the requirements definition report."

College of Computer Studies

Click to edit Master title style

Modeling with Unified Modeling Language (UML)

UML is a popular approach to modeling software and systems. Several UML diagrams provide different viewpoints and blueprints of the system. The **use-case diagram** is one of the UML diagrams.

Functional models describe the business process and the interaction of an information system with its environment. Use-case diagrams can be included in the requirement definition report when defining functional requirements.

Unified modeling language (UML) is a visual modeling language that provides a visual means of developing and documenting object-oriented software and systems.

Components of Use-Case Diagrams

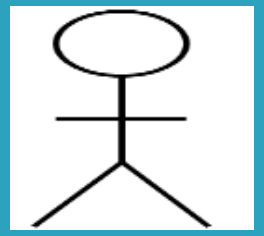
The building components of a use-case diagram include actors, use-cases, subject boundaries, and a set of relationships among actors, actors, and use-cases, and use-cases. The table below describes the components of a use-case diagram showing a set of use-cases and each use-case representing a functional requirement.

TAKE NOTE!

The purpose of a use-case diagram in UML is to summarize and demonstrate the details of functional requirements and how they are used by different users from the point of view of the user of the system.

College of Computer Studies

Click to edit Master title style UML

NOTATION	DESCRIPTION
 Actor/ Role <<actor>> Actor/ Role	Actor • An actor is a person or a system representing the role of someone interacting with the system or software. For example, in a library management system, one of the actors is a Student who interacts with the system. • The stick figure is used when the actor refers to a person type and is labeled with the actor's role. • When an actor represents a role played by a nonhuman, such as an object, another system, or subsystem, a rectangle with <<actor>> keyword and labeled with the role is used to illustrate this type of actor. • Actors can be primary or secondary actors. Primary actors initiate interaction with the system or software. Secondary actors are called upon by the system for support when the primary actor initiates an interaction with a use-case of the system. • A use-case diagram can have one (1) primary actor and can also have several actors of primary and secondary actors. • Actors must be placed outside the subject boundary box. They produce and access data.
Use Case	Use-case • A use-case represents the functionality of a system. • Every use-case must be labeled with a descriptive verb-noun phrase that defines a functional requirement. • The use-case functionality is initiated by an actor. Therefore, every use case is required to be associated with one (1) or more actors. • Use-cases are placed inside the subject boundary box.

College of Computer Studies

Click to edit Master title style

UML

Subject

Subject boundary box

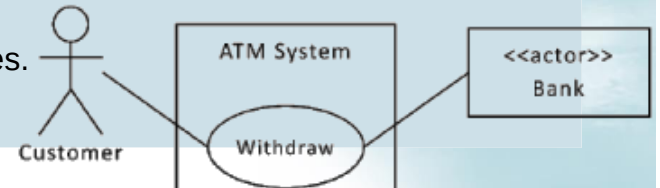
- A subject boundary box represents the system's scope of which a set of use-cases are applied.
- The subject could be a system, software, a module/subsystem such as the login system of a library management system, or an individual business process.
- The subject name must be placed at the top position inside the box.
- All use-cases outside the box are considered outside the scope of that system.

Association relationship

- An association relationship is a line between actor and use-case. This specifies that the actor interacts with the system and uses a specific functionality.
- An actor can be associated with one (1) or more use-cases.
- It is a best practice to place the primary actors on the left side of the subject boundary box, and the secondary actors must be placed on the left side of the box. This visually defines that the primary actors initiate interaction with the system and then the secondary actors provides support or service to the system.

For example, in a simple ATM system, the primary actor Customer and secondary actor Bank are both associated on the Withdraw use-case. This will be easy to understand that the Customer actor initiates the functionality of the Withdraw use-case. Then, the transaction will be sent to the Bank actor for processing the withdraw request of the customer.

- In a use-case diagram, it is important to know which actors are associated with which use-cases.



College of Computer Studies

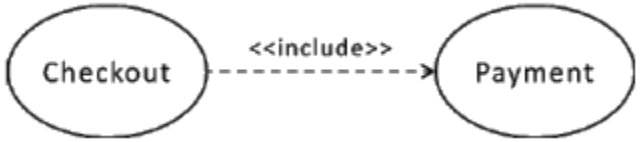
Click to edit Master title style

UML

ask.

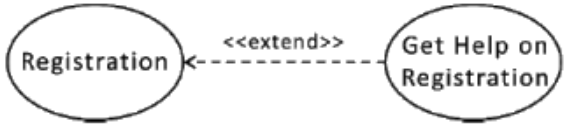
a payment. The

ow is labeled with the



Extend relationship

- An extended relationship indicates optional functionality under a certain use case.
- This defines a base use-case with extending use-case that contains optional behavior.
- For example, a Registration use-case (base) is complete and could be extended with optional Get Help on Registration use-case (extending), since getting help on registration is an optional feature while an actor is registering; hence, the extend relationship is used.



- This is a dashed arrow with an open arrowhead from the extending use-case to the extended (base) use-case. This is labeled with keyword <<extend>>.

College of Computer Studies

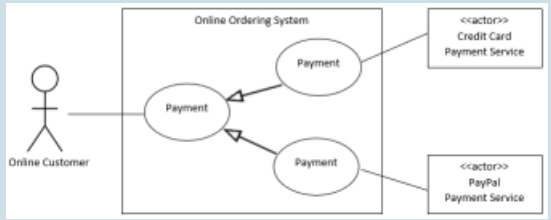
Click to edit Master title style

UML

Generalization relationship

- A generalization relationship represents a specialized use case to a more generalized one. It is similar to generalization between classes. The child use-case inherits the properties and behavior of the parent use-case and may override the behavior of the parent.
- This is used when there are common behaviors between two (2) use-cases and specialized behavior specific to each use-case.

For example, a **Payment** use-case (parent use-case) can be generalized to **Pay by Credit Card** and **Pay by PayPal** use-cases (child use-cases). Then **Payment by Credit Card** and **Payment by PayPal** use-cases inherits the functionality of **Payment** use-case. The **Customer** actor initiates the payment and can select either credit card or PayPal. Both payment methods are associated with their particular payment service. These payment services are in the form of actors that provide services such as verification of a customer's credit card or PayPal.



- This is depicted as an arrow drawn from the child use case to the parent use case.

College of Computer Studies

Identifying the Major Use-Cases

Click to edit Master title style

It is important to identify the use-cases before creating the use-case diagram. It will be easier to create a use-case diagram once the actors, subject boundaries, and uses cases are identified and reviewed.

These are the following steps in identifying major use-cases:

Step 1. **Review requirements definition.** This helps analysts to get a complete overview of the underlying business process being modeled.

Step 2. **Identify the subject's boundaries.** The analysts must identify the subject's boundaries. This helps the analysts in identifying the scope of the system. However, there are changes that may occur during the development process, which will then most likely change the boundary of the subject.

Step 3. **Identify primary and secondary actors and goals.** The analysts must identify the primary and secondary actors that will be involved in the subject and their goals. The primary actors involved with the system come from the list of clients and users. The goals represent the functionality that the subject must provide to the actor. Identifying the tasks that each actor must perform can

facilitate in identifying goals. As actors are identified, and their goals are uncovered, the boundary of the subject will change. Hence, Step 2 is repeated.

Identifying the Major Use-Cases

Click to edit Master title style

Step 4. **Identify business processes and major use-cases.** The analysts must create use-cases for every identified goal of actors. Identifying the major use-cases prevents the users and analysts from forgetting the key business process and helps users explain the overall set of business processes for which they are responsible. It is important to understand and define acronyms and terminologies so that the project team and others (from outside the user group) can clearly understand the use-cases. For example, the Manage Appointments requirement is a major use-case. This requirement included the use-cases for both new patient appointments and existing patient appointments, as well as for canceling the patient appointment. Identifying use-cases is an iterative process, with users often changing their minds about what a use-case is and what it includes.

Step 5. **Review a current set of use-cases.** It is important to review the identified set of use-cases carefully. It may be necessary to split some of them into multiple use-cases or combine some of them into a single use-case. When reviewing the current set of use-cases, a new use-case may be identified.

College of Computer Studies

Click to edit Master title style Creating a Use-Case Diagram

The actual use-case diagram encourages the use of polymorphism or information hiding.The only major parts drawn on the use-case diagram are the system boundary, the use-cases, the actors, and the various relationships between these components. However, when use-case changes during the development process, it could affect the use-case diagram.

There are four (4) major steps in drawing a use-case diagram:

Step 1. **Place and draw use-cases.** Placing and drawing the use-cases on the diagram is the first thing to do. The uses cases are taken directly from identified use cases. The generalization, extend and include relationships can be added to connect the use-cases with relationships. There

is no formal order to the use-cases, so they can be placed in whatever style is needed to make the diagram easy to read and to minimize the number of relationship lines that cross.

Step 2. **Place and draw actors.** The second step is to place the actors in the diagram. To minimize the number of relationship lines that cross on the diagram, place the actors near the use- cases with which they are associated.

Step 3. **Draw subject boundary**. Drawing the subject boundary box will form the border of the subject, separating use-cases from actors. *Step 4*. **Add associations**. The last step is to add associations by drawing lines to connect the actors to the use-cases with which they interact. It is necessary to rearrange the use-cases to help in minimizing the number of association lines that cross.