Kalinga University

Faculty of Computer Science & IT

Course- BCAAIML
Subject- Software Engineering and Testing
Subject Code – BCAAIML503 **Sem- 5th**

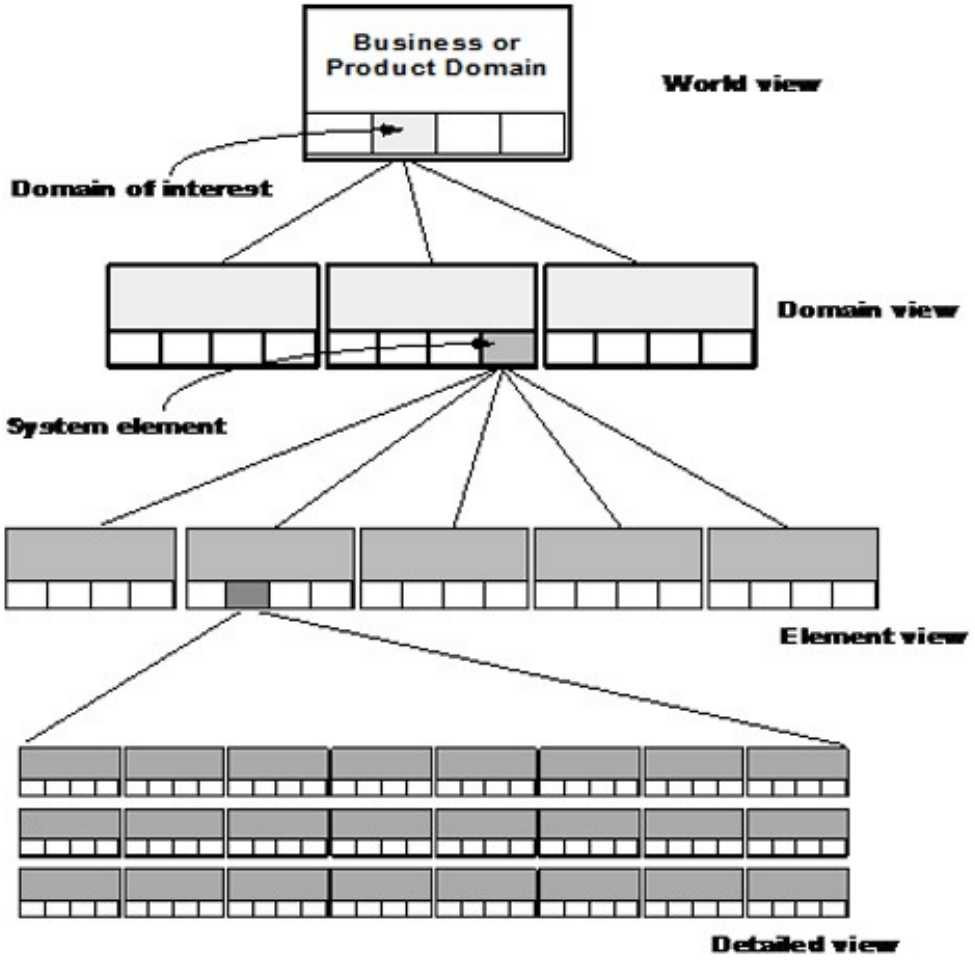
Unit II

COMPUTER-BASED SYSTEMS

The elements of computer-based systems are defined as software, hardware, people, database, documentation, and procedures. A computer-based system makes use of a variety of system elements. Software: programs, data structures, and related work products. Hardware: electronic devices that provide computing capabilities. People: Users and operators of hardware and software. Database: A large, organized collection of information that is accessed via S/w and persists over time. Documentation: manuals, on-line help files. Procedures: the steps that define the specific use of each system element. One complicating characteristic of computer-based system is that the elements constituting one system may also represent one macro element of a still large system. The micro-element is a computer-based system that is one part of a larger computer based system.

THE SYSTEM ENGINEERING HIERARCHY

The key to system engineering is a clear understanding of context. For software development this means creating a "world view" and progressively narrowing its focus until all technical detail is known. In software engineering there is rarely one right way of doing something. Instead designers must consider the tradeoffs present in the feasible solutions and select one that seems advantageous for the current problem. This section lists several factors that need to be examined by software engineers when evaluating alternative solutions (assumptions, simplifications, limitations, constraints, and preferences). Regardless of its domain of focus, system eng. Encompasses a collection of top-down and bottom-up methods to navigate the hierarchy illustrated below:



The system eng. Process usually begins with a "world view." The entire business or product domain is examined to ensure that the proper business or technology context can be established.

The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements (data, S/W, H/W, and people) is analyzed. Finally, the analysis, design, and construction of a targeted system element are initiated.

BUSINESS PROCESS ENGINEERING: AN OVERVIEW

The goal of Business Process Engineering (BPE) is to define architectures that will enable a business to use information effectively.

BPE is one process for creating an overall plan for implementing the computing architecture. Three different architectures must be analyzed and designed within the context of business objectives and goals:

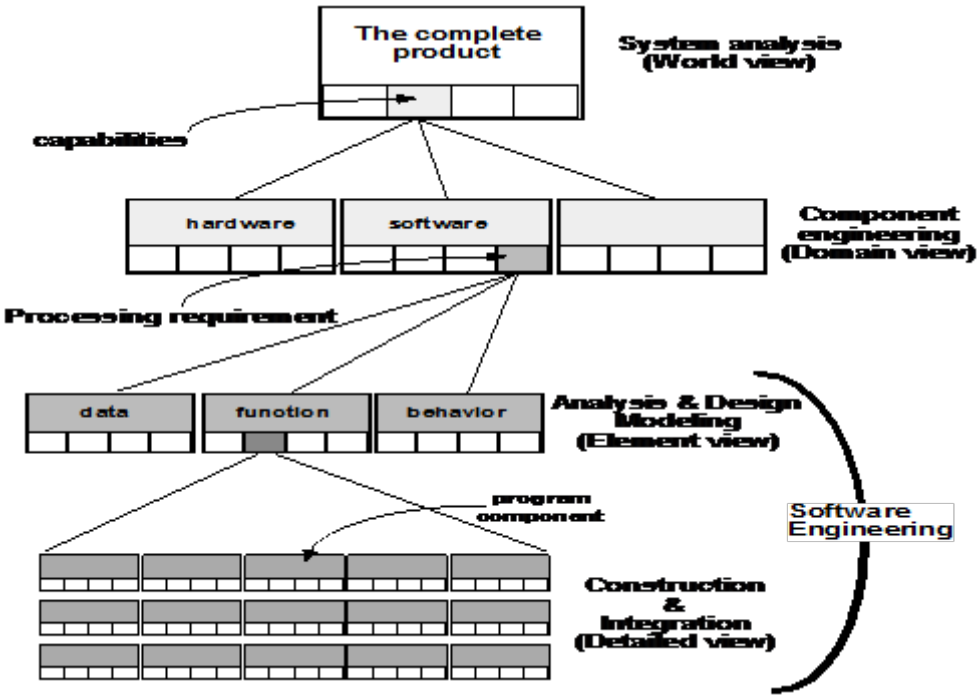
- Data architecture
- Application architecture
- Technology infrastructure

The data architecture provides a framework for the information needs of a business. The building blocks of the architecture are the data objects that are used by the business.

Once a set of data objects is defined, their relationships are identified. A relationship indicates how objects are connected to one another.

The application architecture encompasses those elements of a system that transform objects within the data architecture for some business purpose.

The technology infrastructure provides the foundation for the data and application architectures. The infrastructure encompasses the h/w and s/w that are used to support the applications and data.

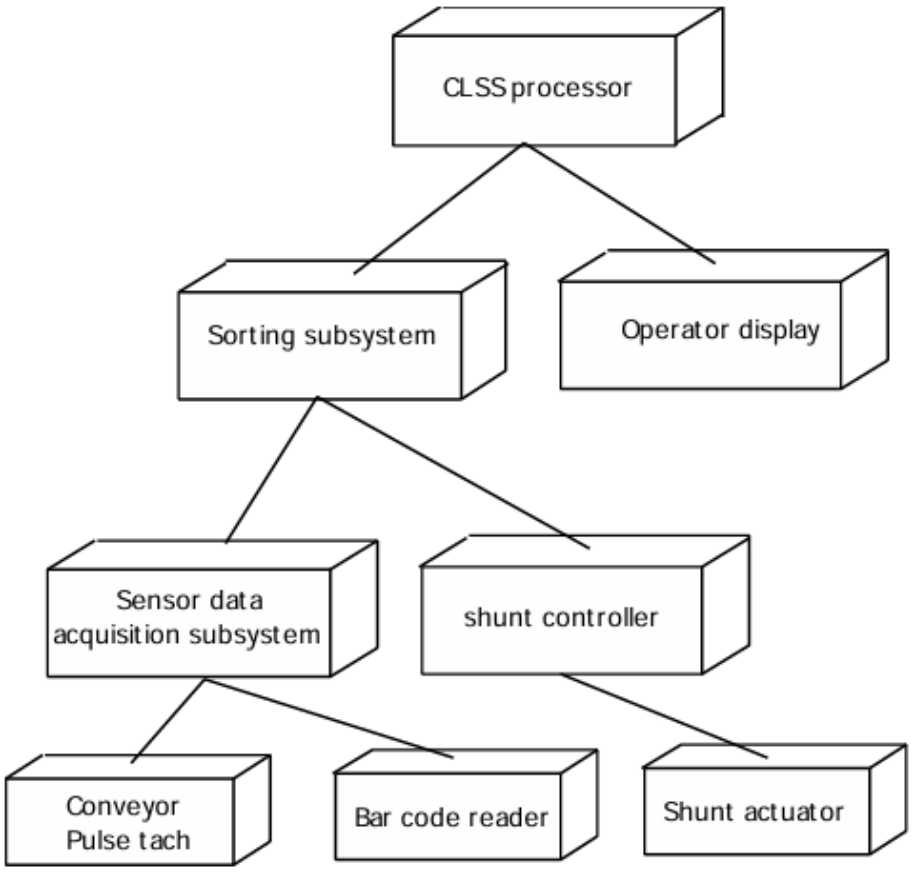


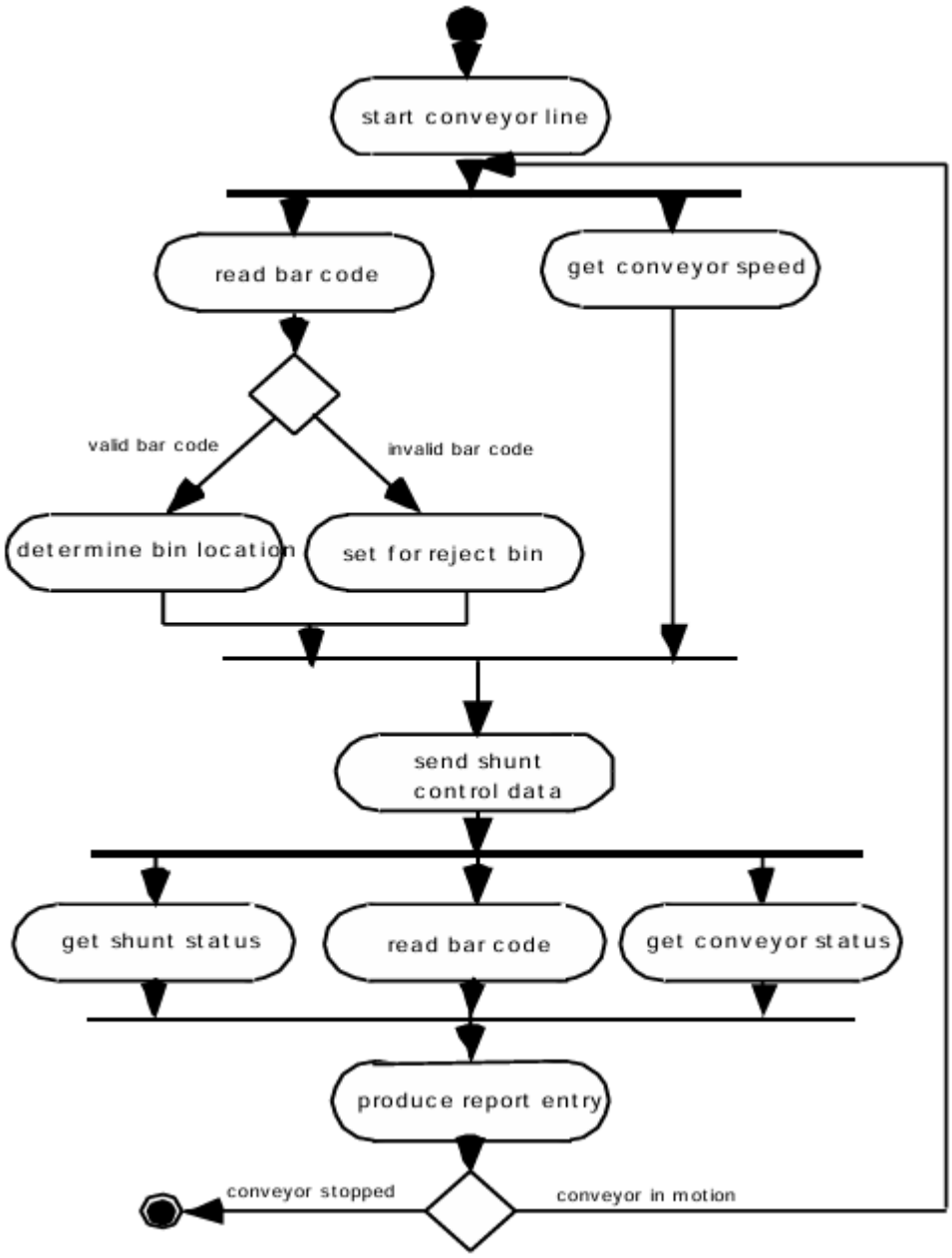
PRODUCT ENGINEERING: AN OVERVIEW

Emphasize that software engineers participate in all levels of the product engineering process that begins with requirements engineering. The analysis step maps requirements into representations of data, function, and behavior. The design step maps the analysis model into data, architectural, interface, and software component designs.

SYSTEM MODELING WITH UML

- In terms of the data that describe the element and the operations that manipulate the data Deployment diagrams
  - Each 3-D box depicts a hardware element that is part of the physical architecture of the system
- Activity diagrams
  - Represent procedural aspects of a system element
- Class diagrams
  - Represent system level elements





Requirements Engineering

- The Problems with our Requirements Practices
- We have trouble understanding the requirements that we do acquire from the customer
- We often record requirements in a disorganized manner

- We spend far too little time verifying what we do record
- We allow change to control us, rather than establishing mechanisms to control change
- Most importantly, we fail to establish a solid foundation for the system or software that the user wants built
- Many software developers argue that
  - Building software is so compelling that we want to jump right in (before having a clear understanding of what is needed)
  - Things will become clear as we build the software
  - Project stakeholders will be able to better understand what they need only after examining early iterations of the software
  - Things change so rapidly that requirements engineering is a waste of time
  - The bottom line is producing a working program and that all else is secondary

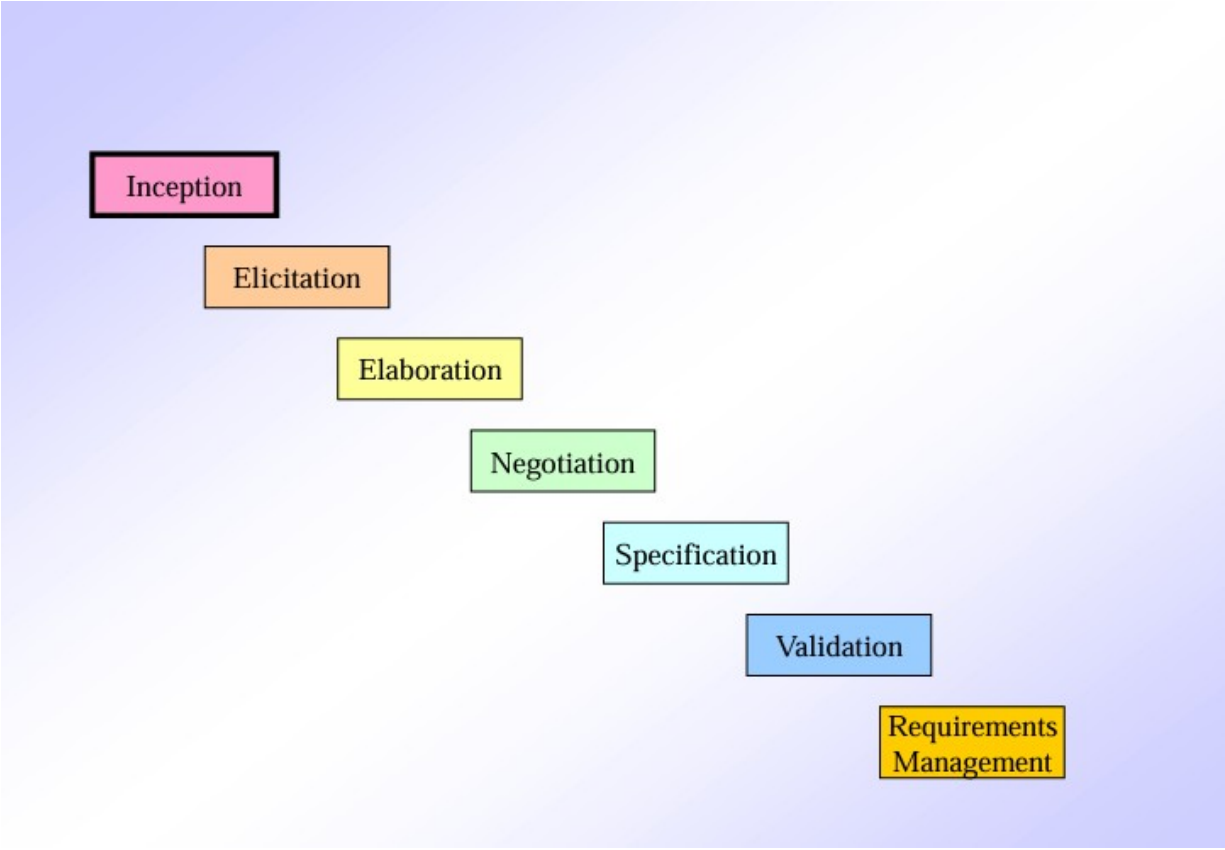
- All of these arguments contain some truth, especially for small projects that take less than one month to complete
- However, as software grows in size and complexity, these arguments begin to break down and can lead to a failed software project

A Solution: Requirements Engineering

- Begins during the communication activity and continues into the modeling activity
- Builds a bridge from the system requirements into software design and construction
- Allows the requirements engineer to examine
  - the context of he software work to be performed
  - the specific needs that design and construction must address
  - the priorities that guide the order in which work is to be completed
  - the information, function, and behavior that will have a profound impact on the resultant design

Requirements Engineering Tasks

- Seven distinct tasks
  - Inception
  - Elicitation
  - Elaboration
  - Negotiation
  - Specification
  - Validation
  - Requirements Management



Inception Task

- During inception, the requirements engineer asks a set of questions to establish…
  - A basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer

- Through these questions, the requirements engineer needs to…
  - Identify the stakeholders
  - Recognize multiple viewpoints
  - Work toward collaboration
  - Break the ice and initiate the communication

Elicitation Task

- Eliciting requirements is difficult because of
  - Problems of scope in identifying the boundaries of the system or specifying too much technical detail rather than overall system objectives
  - Problems of understanding what is wanted, what the problem domain is, and what the computing environment can handle (Information that is believed to be "obvious" is often omitted)
  - Problems of volatility because the requirements change over time
- Elicitation may be accomplished through two activities
  - Collaborative requirements gathering
  - Quality function deployment

Basic Guidelines of Collaborative Requirements Gathering

- Meetings are conducted and attended by both software engineers, customers, and other interested stakeholders
- Rules for preparation and participation are established

- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas
- A "facilitator" (customer, developer, or outsider) controls the meeting
- A "definition mechanism" is used such as work sheets, flip charts, wall stickers, electronic bulletin board, chat room, or some other virtual forum
- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements

Quality Function Deployment

- This is a technique that translates the needs of the customer into technical requirements for software
- It emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process through functions, information, and tasks
- It identifies three types of requirements
  - Normal requirements: These requirements are the objectives and goals stated for a product or system during meetings with the customer
  - Expected requirements: These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them
  - Exciting requirements: These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present

Elaboration Task

- During elaboration, the software engineer takes the information obtained during inception and elicitation and begins to expand and refine it
- Elaboration focuses on developing a refined technical model of software functions, features, and constraints
- It is an analysis modeling task
  - Use cases are developed
  - Domain classes are identified along with their attributes and relationships
  - State machine diagrams are used to capture the life on an object
- The end result is an analysis model that defines the functional, informational, and behavioral domains of the problem

Elements of the Analysis Model

- Scenario-based elements
  - Describe the system from the user's point of view using scenarios that are depicted in use cases and activity diagrams
- Class-based elements
  - Identify the domain classes for the objects manipulated by the actors, the attributes of these classes, and how they interact with one another; they utilize class diagrams to do this
- Behavioral elements
  - Use state diagrams to represent the state of the system, the events that cause the system to change state, and the actions that are taken as a result of a particular event; can also be applied to each class in the system
- Flow-oriented elements
  - Use data flow diagrams to show the input data that comes into a system, what functions are applied to that data to do transformations, and what resulting output data are produced

Negotiation Task

- During negotiation, the software engineer reconciles the conflicts between what the customer wants and what can be achieved given limited business resources
- Requirements are ranked (i.e., prioritized) by the customers, users, and other stakeholders
- Risks associated with each requirement are identified and analyzed
- Rough guesses of development effort are made and used to assess the impact of each requirement on project cost and delivery time
- Using an iterative approach, requirements are eliminated, combined and/or modified so that each party achieves some measure of satisfaction

Specification Task

- A specification is the final work product produced by the requirements engineer

- It is normally in the form of a software requirements specification
- It serves as the foundation for subsequent software engineering activities
- It describes the function and performance of a computer-based system and the constraints that will govern its development
- It formalizes the informational, functional, and behavioral requirements of the proposed software in both a graphical and textual format

Validation Task

- During validation, the work products produced as a result of requirements engineering are assessed for quality
- The specification is examined to ensure that
  - All software requirements have been stated unambiguously
  - Inconsistencies, omissions, and errors have been detected and corrected
  - the work products conform to the standards established for the process, the project, and the product
- The formal technical review serves as the primary requirements validation mechanism
  - Members include software engineers, customers, users, and other stakeholders

Questions to ask when Validating Requirements

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?

Requirements Management Task

- During requirements management, the project team performs a set of activities to identify, control, and track requirements and changes to the requirements at any time as the project proceeds
- Each requirement is assigned a unique identifier
- The requirements are then placed into one or more traceability tables
- These tables may be stored in a database that relate features, sources, dependencies, subsystems, and interfaces to the requirements
- A requirements traceability table is also placed at the end of the software requirements specification.

SYSTEM MODELING

System modeling is an important element of the system eng. Process. The Engineer creates models that:

1. Define the processes that serve the needs of the view under consideration.
2. Represent the behavior of the processes and the assumptions on which the behavior is based.
3. Explicitly define both exogenous and endogenous input to the model.
4. Exogenous inputs link one constituent of a given view with other constituents at the same level of other levels; endogenous input links individual components of a constituent at a particular view.
5. Represent all linkages (including output) that will enable the engineer to better understand the view.

To construct a system model, the engineers should consider a number of restraining factors: Assumptions that reduce the number of possible permutations and variations, thus enabling a model reflect the problem in a reasonable manner.

1. Simplifications that enable the model to be created in a timely manner.
2. Limitations that help to bound the system.
3. Constraints that will guide the manner in which the model is created and the approach taken when the model is implemented.
4. Preferences that indicate the preferred architecture for all data, functions, and technology.

Requirements Elicitation

The process of investigating and learning about a system's requirements from users, clients, and other stakeholders is known as requirements elicitation. Requirements elicitation in software engineering is perhaps the most difficult, most error-prone, and most communication-intensive software development.

1. Requirement Elicitation can be successful only through an effective customer-developer partnership. It is needed to know what the users require.
1. Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system.
1. Requirement Elicitation is a critical part of the software development life cycle and is typically performed at the beginning of the project.
1. Requirements elicitation involves stakeholders from different areas of the organization, including business owners, end-users, and technical experts.
1. The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.
1. Requirements elicitation is difficult because just questioning users and customers about system needs may not collect all relevant requirements, particularly for safety and dependability.
1. Interviews, surveys, user observation, workshops, brainstorming, use cases, role-playing, and prototyping are all methods for eliciting requirements.

Importance of Requirements Elicitation

1. **Compliance with Business Objectives:** The process of elicitation guarantees that the software development endeavors are in harmony with the wider company aims and objectives. Comprehending the business context facilitates the development of a solution that adds value for the company.
2. **User Satisfaction**: It is easier to create software that fulfills end users' needs and expectations when they are involved in the requirements elicitation process. Higher user pleasure and acceptance of the finished product are the results of this.
3. **Time and Money Savings:** Having precise and well-defined specifications aids in preventing miscommunication and rework during the development phase. As a result, there will be cost savings and the project will be completed on time.
4. **Compliance and Regulation Requirements**: Requirements elicitation is crucial for projects in regulated industries to guarantee that the software conforms with applicable laws and norms. In industries like healthcare, finance, and aerospace, this is crucial.
5. **Traceability and Documentation:** Throughout the software development process, traceability is based on well-documented requirements. Traceability helps with testing, validation, and maintenance by ensuring that every part of the software can be linked to a particular requirement.

Requirements Elicitation Activities

Requirements elicitation includes the subsequent activities. A few of them are listed below:

1. Knowledge of the overall area where the systems are applied.
1. The details of the precise customer problem where the system is going to be applied must be understood.
1. Interaction of system with external requirements.
1. Detailed investigation of user needs.
1. Define the constraints for system development.

Requirements Elicitation Methods

There are several requirements elicitation methods. A few of them are listed below:

Requirement Elicitation Techniques

1. Interviews

The objective of conducting an interview is to understand the customer's expectations of the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility. Interviews may be open-ended or structured.

1. In open-ended interviews, there is no pre-set agenda. Context-free questions may be asked to understand the problem.
1. In a structured interview, an agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions

- Brainstorming Sessions is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique

Its objective is to bridge the expectation gap - the difference between what the developers think they are supposed to build and what customers think they are going to get. A team-oriented approach is developed for requirements gathering. Each attendee is asked to make a list of objects that are:

1. Part of the environment that surrounds the system.
1. Produced by the system.
1. Used by the system.

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, the team is divided into smaller sub-teams to develop mini-specifications and finally, a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment

In this technique customer satisfaction is of prime concern, hence it emphasizes the requirements that are valuable to the customer.

3 types of requirements are identified:

- **Normal requirements:** In this the objective and goals of the proposed software are discussed with the customer. For example - normal requirements for a result management system may be entry of marks, calculation of results, etc.
- **Expected requirements:** These requirements are so obvious that the customer need not explicitly state them. Example - protection from unauthorized access.
- **Exciting requirements:** It includes features that are beyond customer's expectations and prove to be very satisfying when present. For example - when unauthorized access is detected, it should back up and shut down all processes.

5. Use Case Approach

Use Case technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.

The components of the use case design include three major things - Actor, use cases, and use case diagram.

1. **Actor:** It is the external agent that lies outside the system but interacts with it in some way. An actor may be a person, machine, etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.

- **Primary actors:** It requires assistance from the system to achieve a goal.
- **Secondary actor:** It is an actor from which the system needs assistance.

1. **Use cases:** They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.

1. **Use case diagram:** A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.

- A stick figure is used to represent an actor.
- An oval is used to represent a use case.
- A line is used to represent a relationship between an actor and a use case.

The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.

Steps of Requirements Elicitation

Following are the Steps of Requirement Elicitation

Steps of Requirement Elicitation

1. Identify Stakeholders

The first step is to figure out who all the key players are in the project. These are the people who will either use the product, help develop it, or be impacted by it. This could include users, developers, project managers, and customers. It's important to get everyone involved early on, so you gather all the necessary input and perspectives when defining the requirements.

2. Gather Requirements

Once you've identified the stakeholders, the next step is to gather their requirements. This means understanding what they need from the system or product. Requirements can be functional (what the system should do) or non-functional (how it should perform). You can use different methods like interviews, surveys, or group discussions to collect all these needs.

3. Prioritize Requirements

Not all requirements are equal in terms of importance. After gathering them, you'll need to prioritize which ones should be addressed first. This helps ensure that the most important features are built into the system first, especially when resources (like time and budget) are limited. A common approach is to categorize them into things that are "Must-have," "Should-have," "Could-have," and "Won't-have."

4. Categorize Feasibility

After prioritizing, the next step is to assess how realistic each requirement is. You'll categorize them into three groups:

- **Achievable**: These are requirements that are realistic and can be done within the available resources (time, budget, etc.).
- **Deferred**: These requirements are important but can't be implemented in this phase. They'll be put on hold for now with a clear reason why.

- **Impossible**: These requirements can't be done because of technical or other limitations, and they should be removed from the list.

Features of Requirements Elicitation

1. **Stakeholder engagement:** Requirements elicitation involves engaging with stakeholders such as customers, end-users, project sponsors, and subject-matter experts to understand their needs and requirements.
1. **Gathering information:** Requirements elicitation involves gathering information about the system to be developed, the business processes it will support, and the end-users who will be using it.
1. **Requirement prioritization:** Requirements elicitation involves prioritizing requirements based on their importance to the project's success.
1. **Requirements documentation:** Requirements elicitation involves documenting the requirements clearly and concisely so that they can be easily understood and communicated to the development team.
1. **Validation and verification:** Requirements elicitation involves validating and verifying the requirements with the stakeholders to ensure they accurately represent their needs and requirements.
1. **Iterative process:** Requirements elicitation is an iterative process that involves continuously refining and updating the requirements based on feedback from stakeholders.
1. **Communication and collaboration:** Requirements elicitation involves effective communication and collaboration with stakeholders, project team members, and other relevant parties to ensure that the requirements are clearly understood and implemented.
1. **Flexibility:** Requirements elicitation requires flexibility to adapt to changing requirements, stakeholder needs, and project constraints.

Advantages of Requirements Elicitation

1. **Clear requirements:** Helps to clarify and refine customer requirements.
1. **Improves communication:** Improves communication and collaboration between stakeholders.
1. **Results in good quality software:** Increases the chances of developing a software system that meets customer needs.

1. **Avoids misunderstandings:** Avoids misunderstandings and helps to manage expectations.
1. **Supports the identification of potential risks:** Supports the identification of potential risks and problems early in the development cycle.
1. **Facilitates development of accurate plan:** Facilitates the development of a comprehensive and accurate project plan.
1. **Increases user confidence:** Increases user and stakeholder confidence in the software development process.
1. **Supports identification of new business opportunities:** Supports the identification of new business opportunities and revenue streams.

Disadvantages of Requirements Elicitation

1. **Time-consuming:** It can be time-consuming and expensive.
1. **Skills required:** Requires specialized skills and expertise.
1. **Impacted by changing requirements:** This may be impacted by changing business needs and requirements.
1. **Impacted by other factors:** Can be impacted by political and organizational factors.
1. **Lack of commitment from stakeholders:** This can result in a lack of buy-in and commitment from stakeholders.
1. **Impacted by conflicting priorities:** Can be impacted by conflicting priorities and competing interests.
1. **Sometimes inaccurate requirements:** This may result in incomplete or inaccurate requirements if not properly managed.
1. **Increased development cost:** This can lead to increased development costs and decreased efficiency if requirements are not well-defined.

Software Prototyping

A software prototype is a simulation of how the actual project will look, work, and feel and would not have an exact logic involved in the software development phase. Software development team members use it for designing user feedback and user testing. And they come in various levels of sophistication. Software prototyping refers to the process that starts by creating an idea, sketching it, and making it into a clickable prototype that mimics real software.

The prototype can offer simulation for the entire software or mobile application as per the requirement of the user. Basically, prototyping in software development is just like a scale building model that is utilized in designing architecture. In that case, creating a sophisticated prototype is necessary as it enables making a complex project look easy. It enables the customers to see the exact plan of the software development and by having a look at that, they can provide feedback to the software designer.

The best part of a software prototype is that it is cheaper and easier as changes can be made in the middle of the software development process instead of making changes after its completion. Therefore, while creating software, an engineer prefers to develop an initial prototype that he/she can use to gather feedback from the project owner.

Why Do You Need A Software Prototype?

When it comes to creating software for the client from the very beginning, the process starts from the initial planning phase. The software development team comes with an application idea that is very well thought out so that the implementation can be easy. The team follows the software development life cycle (SDLC) process to have a clear and straightforward approach among the team members.

This process starts with developers gathering all the software-related information from the client and understanding the client's requirements. The next step is to start the designing process. It offers details of what the software's end-user interface will look like. After that comes the actual coding part of the system. It is the most lengthy part of the software development life cycle. Later, the team checks the code and fixes the bugs if there are any. And the final step to do is to launch it. And in this entire process, prototyping software can help by cutting down the overhead costs.

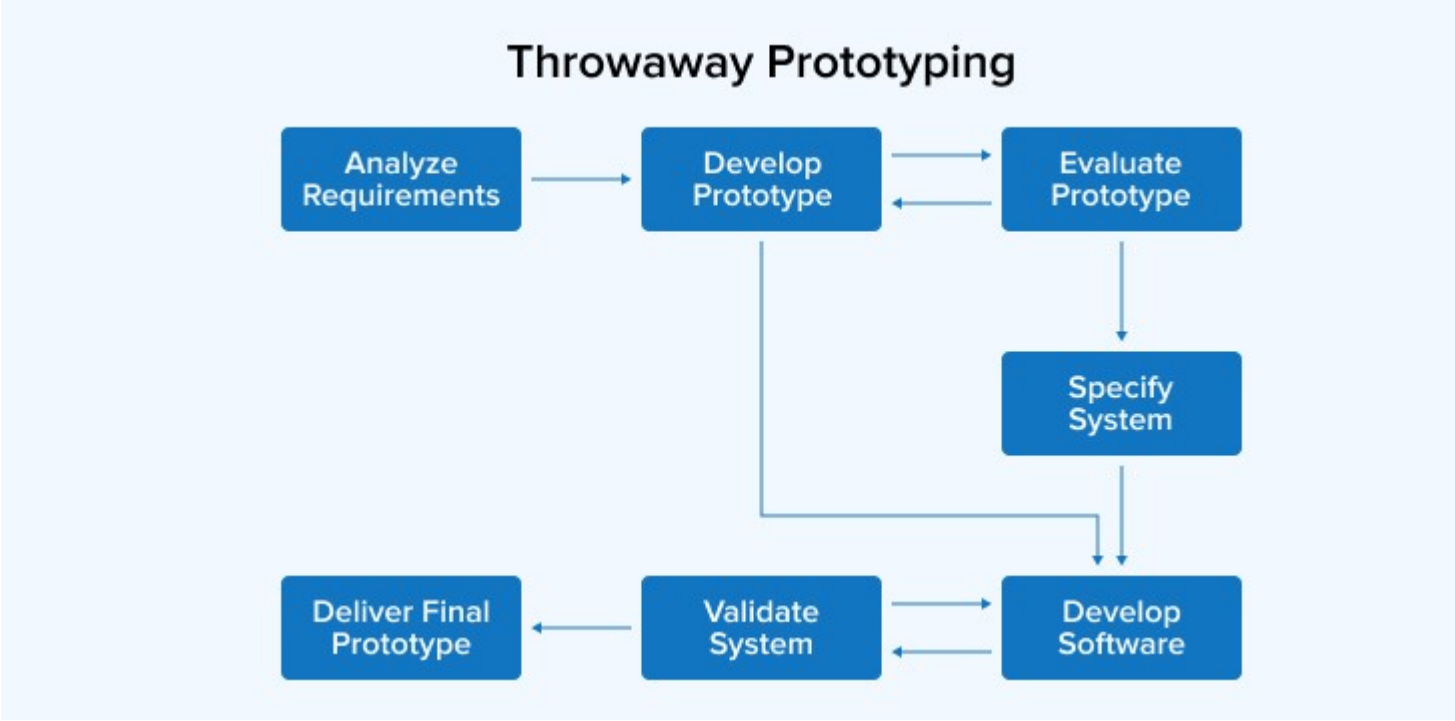
Basically, software prototypes help software developers to visualize the final product before the software development company starts investing resources into it. It is not an exact replica of the software but it is a quick mock-up that offers a rough idea about the system's functioning and appearance to the software engineering team. The team can see the prototype and can decide whether the idea of this project is worth the investment or not. Besides this, as prototypes offer a clear idea about the system, the end result can be as user-friendly as possible.

Types of Prototyping Models

Here are some of the top types of prototyping models –

1. Throwaway Prototyping

Rapid throwaway or rapid prototyping is one the most important type of prototyping models. It is based on the preliminary requirement of the software applications. The throwaway prototype can be quickly developed by the team to show how the initial requirements of the system will look visually. In this model, the feedback from the customers helps in driving changes in the requirements and after the user feedback is received, the prototype is again created to meet the client's needs.

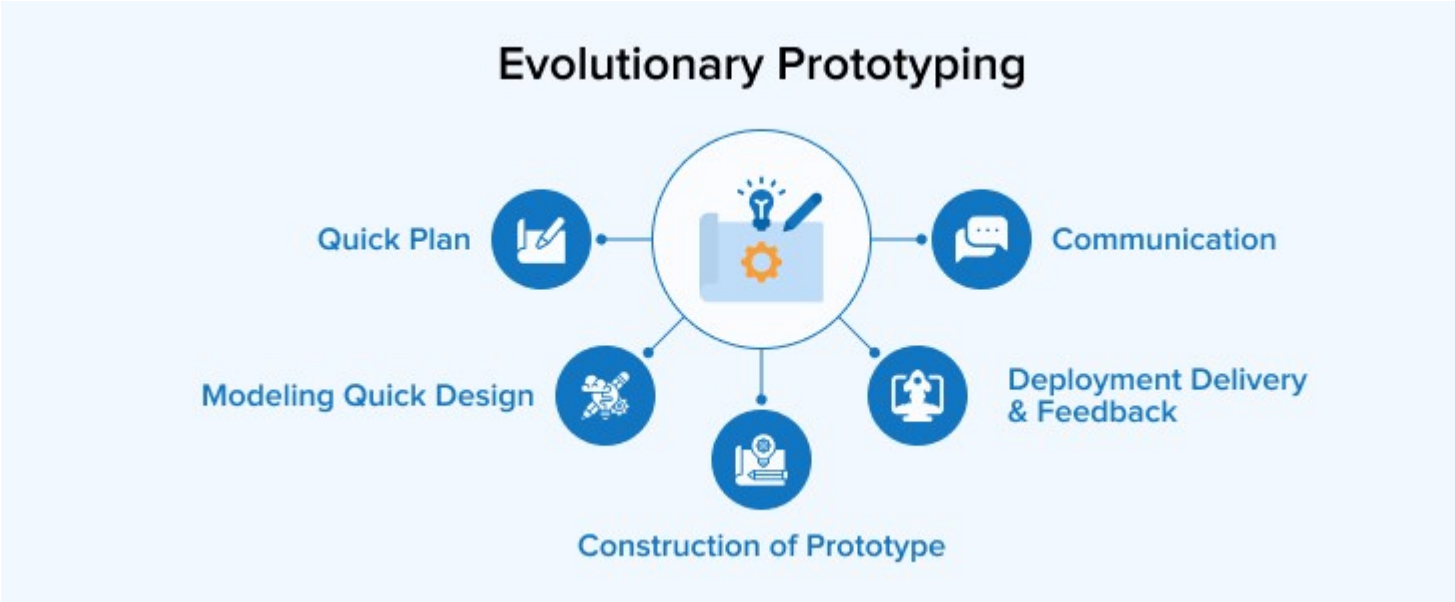


Basically, in this approach, the prototype created by the team will be discarded and won't become a part of the final accepted prototype. This model is used by the software product designing team to explore ideas and get instant feedback from the clients.

2. Evolutionary Prototyping

Another type of prototyping is an evolutionary prototype. Evolutionary prototypes are created by the team in an incrementally refined manner after accepting the customer's final feedback. This

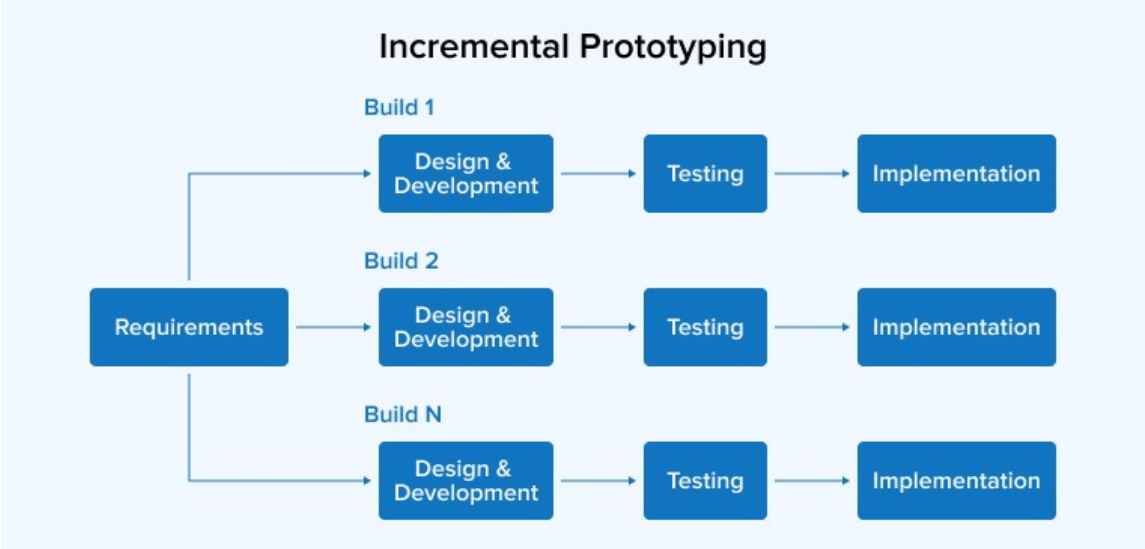
type of prototyping enables the team to save a lot of development time and effort. And the main reason behind it is that the development of a prototype from scratch for every different interaction can be very frustrating.



The design team works on prototypes like this when the technology that is going to be used in the project is new and not well understood by the team. It is also the perfect model when it comes to working on a complex project where all the features must be checked at least once. Basically, evolutionary prototyping in software development is useful when the software requirement is not understood perfectly or it is not stable at the initial stage. Besides, the screens (user interface) also have actual code behind them. The user can see and interact with the prototype as if it were the actual product. Over time and multiple feedback cycles, the prototype may have more advanced functionality added to it as needed by the client. The process thus results in the finished product.

3. Incremental Prototyping

Incremental is a type of prototyping that enables the division of the final product into smaller prototypes and then they are developed individually. This means that different prototypes can be merged into a single software. This approach is used to reduce the feedback time between the application development team and the user.

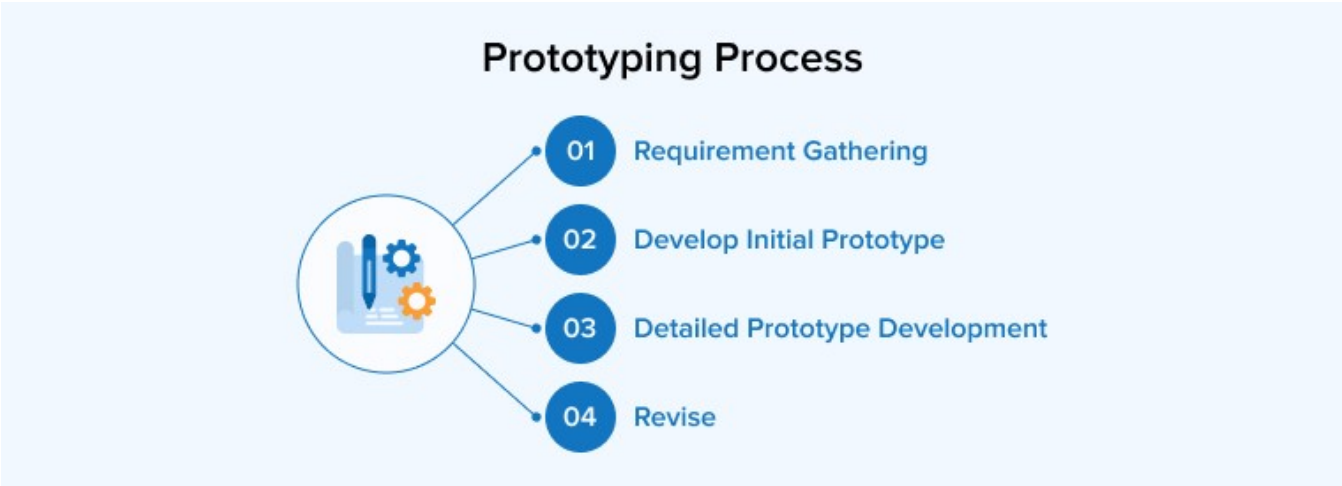


4. Other Types

Besides the above-listed prototypes, there are some other software prototyping techniques. They are of different types as per the level of software details one wants to see in the prototype. And when a very basic or low-effort prototype is created, it is called a low-fidelity prototype. The best example of this is a paper prototype. The designing team can also create a prototype in a detailed form, and it is known as a high-fidelity prototype. Basically, creating a prototype depends on the software requirements, the software developers or designers, and how the experts use the client's user interface design and software prototyping tools to represent the system.

The Prototyping Process

Here are the steps that every software developer must follow while going through the prototyping process.



1. Requirement Gathering

When it comes to the prototyping process, the very first step is to identify the problem of the software and come up with a complete version of the software solution. In this, the developing team must be very clear about what activities the application will be used on the platform. And for this, the developers must ask the project owner for extensive details about the features and functionalities. And here the owner can give his ideas regarding the potential software design. After understanding the project's type and its requirements, the developers can create the prototype.

Before starting the creation of the prototype, the developer must see to it that he has all the required details. As well as per the requirement analysis, he can specify all the software details in the prototype.

2. Develop Initial Prototype

The earliest prototypes are created by focusing on the general arrangement of the appearance of the software application. Besides this, the client also describes the functionality of the system in the initial stage by the client. It can be in the form of on-paper wireframes which is low-fidelity prototypes. Here the outline of the application is informally outlined and a few basic functions will also be highlighted.

3. Detailed Prototype Development

If the developers feel like they need to create a more detailed functional prototype of the software for their own use and for the client's review, they can develop a detailed version of the software design and coding. This can help us to understand the business processes by the design and development team. And for this, they can use advanced design and software prototyping tools to develop a comprehensive architecture.

4. Revise

The last step in the software prototyping process is to make revisions to the prototypes created after taking the client's feedback & requirements into consideration.

Advantages Of Software Prototyping

Some of the major advantages of prototyping software applications are –

1. Get Started Right Away

Developing a software application is not always an easy task, it can be quite daunting especially when the software is complex and the team has to think about every single thing the client requires. In the beginning, the engineers can share a few ideas with the client and after the approval, the designers can get start working by creating a rough that can be refined later if required.

2. Reduces Risk

One of the most important benefits of software prototyping is reducing risks. If the developer wants to avoid the scenario from the start and reduce the risk of software failure, he must implement software prototyping. This concept enables the developers to offer the perfect solution as per the client's feedback.

3. Offers Visual Guide

Software prototyping comes with the best feature known as a visual guide. It enables the developers to see quick results. The best part about it is that the clients can also get an easy-looking overview of the system that the hired team has designed.

The basic prototype of the system enables both the clients and developers to get the perfect idea of the system. While the high-fidelity prototype is an approach that enables one to see what exactly the software solution will look like after it will be developed. A high-fidelity prototype allows one to have an interaction with it by going through different screens, making sure the application flows smoothly & perfectly, and clicking up buttons.

4. Allows Detecting Errors at the Beginning

With the help of software prototyping, developers understand the issues in the system very easily. They can detect errors in the product concept. And because of the early detection, problematic issues can be solved efficiently and quickly. This means that errors cannot disrupt the development process as the errors are detected and eliminated from the actual product. It also enables in reducing the cost and time of implementing changes.

5. Save Time and Money

When new software is implemented in any organization, learning and understanding it and its functionalities can be difficult for the employees. Even when the employees start learning more about the system, it is not necessary that the team will be happy with it. The company should hire a development company that creates prototypes and takes feedback on them before starting to work. Through this, the business owner can give feedback and make the system develop as per the requirements.

This means that prototyping is one of the best things to do if one wants to identify problems early. This is beneficial for both developers and business owners. It helps in saving a lot of time and money.

6. Improves Client's Involvement

Software prototyping also has the capability to enable the customers to take part in developing the system. And this can be beneficial for both development team members and clients. The reason behind it is that it enables the clients to share their critiques and opinions. While it allows the development team to be fully aware of customers' needs.

Disadvantages Of Software Prototyping

Some of the disadvantages of software prototyping are –

1. Insufficient Analysis

When a developer only limits his focus on the prototype, it can distract him from analyzing the entire project. One can overlook the better solutions, conversion of limited prototypes, and incomplete specifications.

2. User Confusion

Some of the clients feel that the prototype is a finished project. They see the rough version of the system and feel that this is it, the final software product is just a polish away. And this also makes the clients wrongly perceive the prototype to be an accurate model.

3. Developer Misunderstanding of User Objectives

For every software development project to be successful, both customers and developers have to be on the same page. This means that when clients give a list of all required features and the clients

show that in the prototype, it can create a conflict in the team. The main reason behind it is that all the suggested features might now be perfect for the software.

4. Excessive Development Time

Prototypes are something that is created so that the development process can be followed quickly. If the developer spends a lot of time designing a complex prototype, the project can run late and this can also cost a lot of money.