Software Engineering

CSIT- 6th Semester

Prepared by: Er. Rupak Gyawali, 2024

Unit 2: Software Processes

Software Process; Software Process Models (Waterfall Model; Incremental Development; Integration and Configuration); Software Process Activities (Software Specification, Software Design and Implementation; Software Validation; Software Evolution); Coping with Change (Prototyping, Incremental Delivery); Process Improvement

Software Process:

- A software process is a series of steps that helps create a software product. This can include building software from the scratch using programming languages like Java or C.
- However, many business applications are now made by improving existing systems or by using and combining ready-made software instead of starting from scratch.
- There are many different software processes but all must include four activities:
    1. **Software Specification**: Define what the software should do and any limits on how it works.
    2. **Software Design and Implementation**: Create the software based on the specifications.
    3. **Software Validation**: Check that the software works as the customer wants it to.
    4. **Software Evolution**: Update and improve the software to keep up with changing customer needs.
- Example of Software Process: Waterfall, Agile, Spiral, V-Model, Incremental model, RAD etc.

Software Process includes core activities:

- **Products:** The results of process activities. For example, After developing the software, a user manual may be created as a product to guide users on how to install and use the software effectively.
- **Roles:** These show who does what in the process. Common roles include project manager, configuration manager, and programmer.
- **Pre-conditions:** Things that must be true before starting a process activity. For example, all requirements should be approved by the customer before the design work begins.
- **Post-conditions:** Things that should be true after a process activity is completed. For example, after finishing the architectural design, the models should be reviewed.

Software Process:

Software processes are complicated and depend on people making decisions. Every organization has its own way of developing software because there isn't a perfect process.

- **Structured vs. Flexible Processes:** Some systems, like critical ones (e.g., medical software), need a very organized approach, while business systems, which change quickly, work better with a flexible method.
- **Plan-Driven vs. Agile:** Software processes can be divided into plan-driven, where everything is planned in advance, and agile, where planning is done in small steps and can easily adapt to changes.
- **Improving Processes:** While there's no perfect process, many organizations can improve theirs. This can involve reducing different methods within the organization to make communication easier, cut down on training time, and use automation better.

Software Process Model:

a) The Waterfall Model:

- The waterfall model is one of the earliest software development models, created by Royce in 1970.
- It's called the "waterfall" because each step flows into the next, like a waterfall. It is a plan-driven approach, so everything is planned and scheduled before work starts.
- Although the original developer of the waterfall model, W. W. Royce, called for feedback between phases in the waterfall, this feedback came to be ignored in implementation (Martin, 1999).

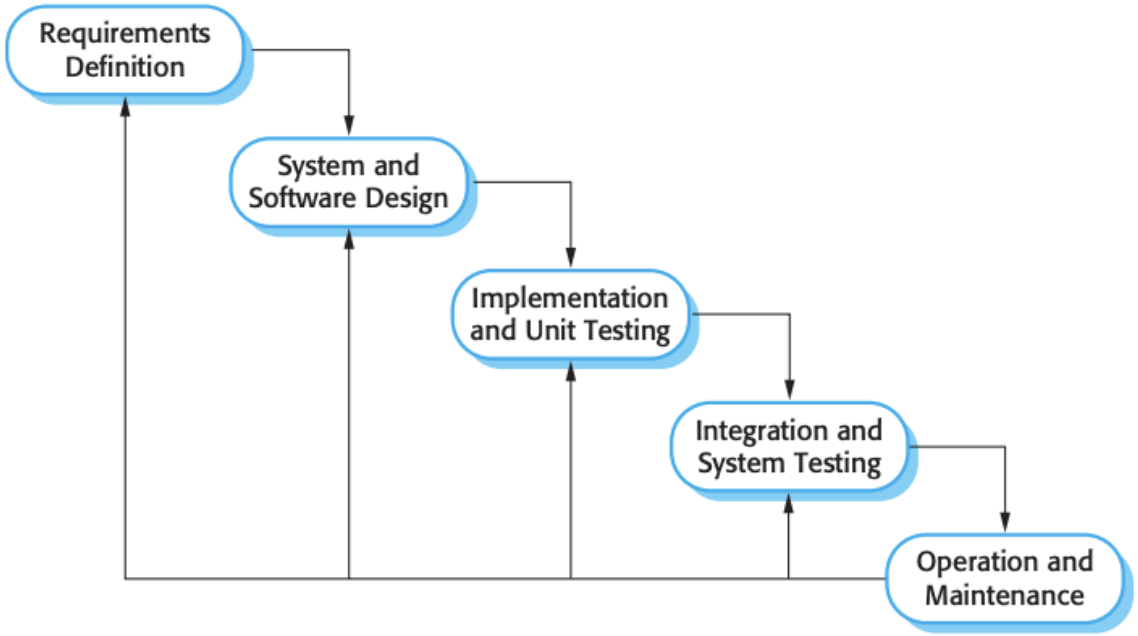


Figure 2.1 The waterfall model

Here's a simple breakdown of each phase:

a) The Waterfall Model:

- **Requirements Analysis and Definition:** Work with users to understand and list out what the system should do, what goals it should meet, and any limits. This list becomes the official guide for building the system.
- **System and Software Design:** Plan how the requirements will be split between hardware and software, and sketch out the main structure of the software, showing how different parts connect.
- **Implementation and Unit Testing:** Write the actual code for the software and test each part on its own to ensure it works correctly.
- **Integration and System Testing:** Put all the pieces together and test the full system to make sure it does everything it's supposed to. Once it passes, it's ready to go to the customer.
- **Operation and Maintenance:** After the system is live, it's used in real situations. Any missed bugs are fixed, and updates or improvements are made over time as needed. This phase usually lasts the longest.

- Each phase in the software process creates documents that need approval before moving on.
- While ideally, each phase should finish before the next starts, in practice, they overlap, and feedback from one phase may affect previous phases.
- For instance, design might reveal issues in the requirements, or coding might highlight design flaws, requiring updates to earlier documents.
- Since making changes can be costly, teams often limit changes by "freezing" certain stages, like requirements. This helps avoid extra work but might mean the final system isn't exactly what users wanted.
- In the final stage (operation and maintenance), the software is used, and issues or missing features are discovered, requiring updates. These updates may involve revisiting earlier phases to improve the software and keep it useful.
- The waterfall model follows a step-by-step approach with clear documents at each stage, making it easy for managers to track progress.
- The waterfall model is not very flexible and doesn't handle changes in requirements easily, so it's best for projects where the needs are clear and unlikely to change.

a) The Waterfall Model contd…:

- A variation of the waterfall model, called formal system development, uses mathematical methods to ensure that the system follows its specifications. This approach is helpful for critical systems where safety, reliability, or security is essential. By building a mathematical model and refining it into code, developers can show that the system meets strict safety or security standards. However, because this method is complex and costly, it's mostly used for specialized, high-stakes projects rather than general software development.

Problems with Traditional Waterfall Approach

- **Step-by-Step Process:** The waterfall model goes through each stage (like planning, Analysis, design, Implementation and maintenance) one after the other. But if there's a problem in one stage, it's hard to go back and fix it without redoing everything.
- **Not Flexible:** Once you finish a stage, it's tough to make changes. If you need to change something later on, it can be a big hassle and might take a lot of time and money.
- **Late Feedback:** People only give feedback on the product near the end of the process. By then, it's hard to make any big changes, so you might end up with something that doesn't meet everyone's needs.
- **High Risk:** There's a big chance the project might fail, especially if the requirements keep changing. Without checking things early on, you might deliver a product that doesn't work well for users.
- **Takes a Long Time:** Since everything happens one after the other, it can take a while to finish the project. This can make people impatient and frustrated.
- **Not Much Teamwork:** Each team works on their part separately, which can make it hard to share ideas and work together effectively.

When to use Waterfall Model?

- **Requirements are clear and unlikely to change:** You know exactly what's needed from the start.
- **Project is simple and well-defined:** Each stage can be planned and completed in order.
- **Strict deadlines:** A structured approach makes it easier to track progress and meet deadlines.
- **Quality and thorough documentation are essential:** Each phase has clear documentation, which helps maintain quality control.

a) The Waterfall Model contd…:

Advatages of Waterfall model:

-Easy to understand and Implement.

-Documentation is produced in each phase.

-SQA group carefully checks the system after end of each phase or testing is performed in each stage.

-Identifies milestones or deliverables.

Dis- advantages of Waterfall model:

- Difficult to change the requirement if the process in underway. One phase must be completed to move on to the next phase.
- Uni-directional.
- Confusion due to unclear requirements.
- Client approves only after final stages.
- Difficult to integrate risk management.

b) V-Model

The V-model is a variation of the waterfall model that shows how quality assurance (testing and validation) relates to the development stages.

Key Points of the V-Model:

- **Two Sides of the V:**
- **Left Side:** This side represents the development stages. Here, basic requirements are gradually refined into detailed designs and coding.
- **Right Side:** After coding, the team moves up this side to perform testing. Each test checks that the previous stages were done correctly.
- **Verification and Validation:** As the team moves down the left side, they create models of the software. When they move up the right side, they validate (check) each of these models through testing.
- **Visualization:** The V-model visually highlights the importance of testing and quality assurance at each stage of development, ensuring that each step is correctly implemented before moving to the next.

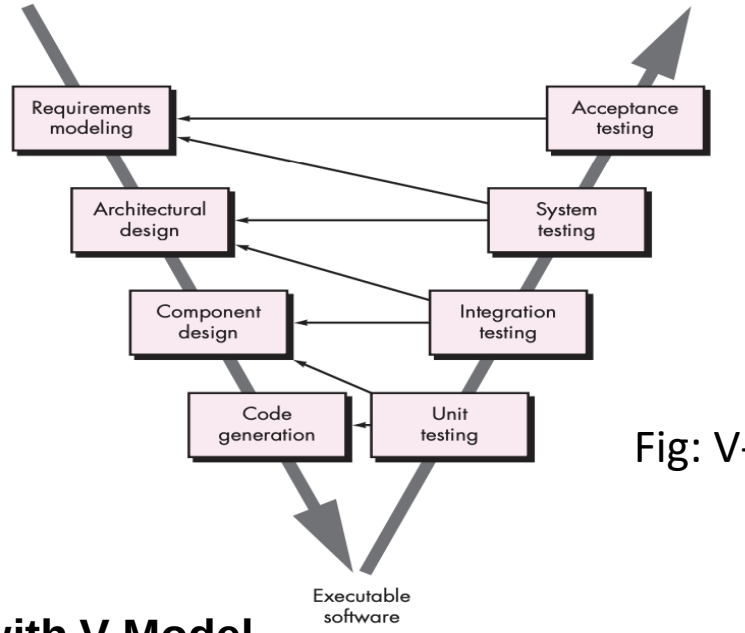


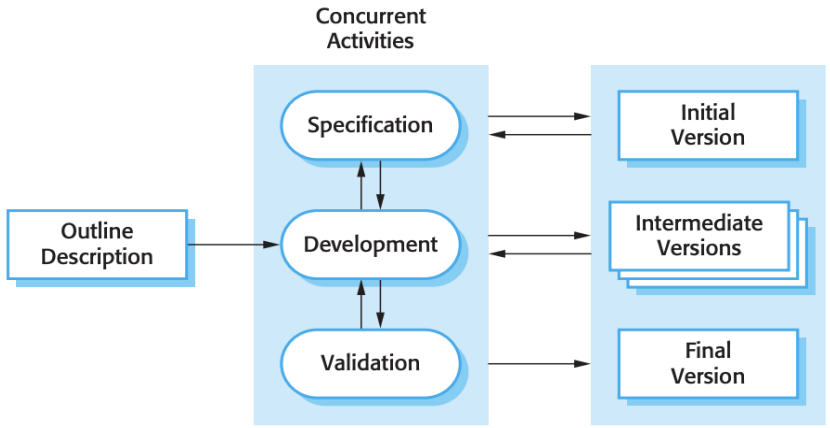
Fig: V- Model

Problems with V-Model

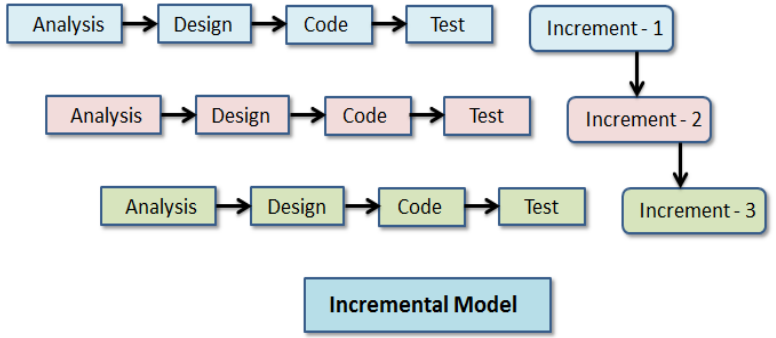
- **Inflexible:** Like the waterfall model, the V-model is Inflexible. Once you start a phase, it's difficult to go back and make changes if you find a problem later.
- **Late Testing:** Testing only happens after development. If issues are discovered during testing, it can be costly and time-consuming to fix them.
- **Assumes Clear Requirements:** The V-model assumes that all requirements are clear from the start. If requirements change or are misunderstood, it can lead to significant problems.
- **High Initial Costs:** The emphasis on thorough documentation and testing can lead to high initial costs, which may not be feasible for all projects.

c) Incremental Development:

- Incremental development involves creating an initial version of a system, letting users test it, and then gradually improving it through multiple versions until it meets user needs as shown in figure.



- Instead of separating tasks like planning (Specification), coding (Development), and testing (validation), these are done together with fast feedback between them.
- Incremental development is popular in agile methods and works well for business, e-commerce, and personal projects. Each new version adds important features first, so users can give early feedback. If changes are needed, only the latest version is updated, and new features can be added in future versions.



Incremental Model

- At each increment, small working part of the product is produced.
- At each increment, user is provided with small working module so that user can see the module and change his requirements.

Eg: Suppose you need to deliver a complete wordprocessor system at the day of deadline. It is impossible to provide a complete system at the deadline day.

- So at the 1st release → Provide file management, document production and editing functionalities.
- At next increment → provide advanced editing tools, advanced file editing capabilities.
- At further increment → provide grammar checking, mailing and so on.

c) Incremental Development:

Benefits of Incremental Model over the waterfall model:

1. **Lower Cost of Changes**: When customers need changes, it's cheaper and faster to make updates since there's less analysis and paperwork compared to the waterfall model.
2. **Easier Customer Feedback**: Customers can see working parts of the software sooner, making it easier for them to give feedback. This is much clearer than reading design documents.
3. **Faster Delivery**: Parts of the software can be delivered and used earlier, even if it's not fully complete. This means customers get value from the software sooner.

Incremental development is now the most popular way to build applications. It can be plan-driven, agile, or a mix of both:

- **Plan-driven**: The stages of development are planned in advance.
- **Agile**: Early stages are planned, but later stages change based on progress and what customers need.

From a management perspective, incremental development has two main challenges:

1. **Less Visibility:** It's hard for managers to track progress without regular, official documents for each version. Making these documents for every version can get costly.
2. **Structure Degradation:** As more updates are added, the software's structure can get messy. Without regular improvement (refactoring), the system becomes harder and more expensive to update.

Advantages of Incremental:

- Customer can see Product earlier.
- Lower Risks of overall project failure.
- Lower cost to change customer Requirements.
- Eaiser to test and debug during a small iteration.

Some Dis-Advantages:

- Increment should be relatively small (20K Lines of Code).
- Needs Good planning and Design.

When to Use Incremental Model?

- **Requirements are unclear or likely to change**: Building in small steps lets you adjust as you learn more about what's needed.
- **Early delivery is important**: Customers get basic functions sooner, and more features are added over time.
- **Project is complex**: Breaking it into smaller parts makes it easier to manage and test.
- **Budget or time is limited**: You can deliver parts of the system within the constraints and add more later.

d) Integration and Configuration/ Reuse oriented SE:

- In reuse-oriented software engineering, parts of existing software are reused in new projects. Developers often look for code or designs they already know about, modify them as needed, and add them to their new system.
- Reuse happens in most software projects, some development methods now focus specifically on reusing software components.
- Reuse methods depend on a large set of reusable software parts and a framework to integrate together. Sometimes, the reusable parts are ready-made software systems, like a word processor or spreadsheet, which offer specific features.
- In reuse-based development, the steps include defining requirements, finding reusable components, integrating these parts, and validating the system. The key difference is that you focus more on using existing parts instead of building everything from scratch.

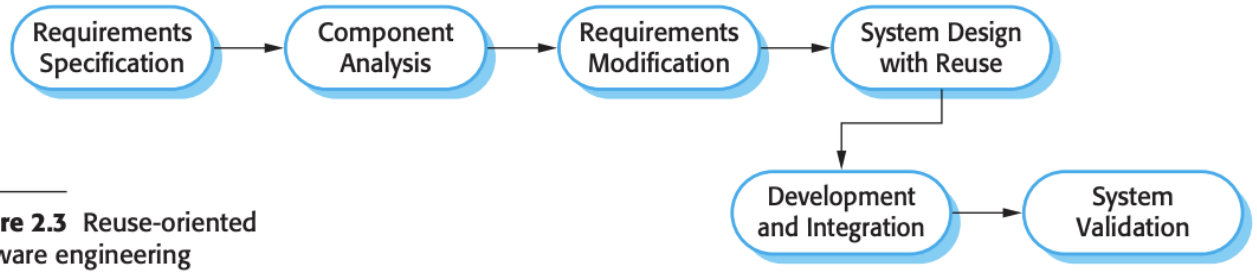


Figure 2.3 Reuse-oriented software engineering

- The reuse-oriented software engineering model is sometimes referred to as **integration and configuration.** This term emphasizes the focus on integrating existing software components and configuring them to meet specific requirements, rather than developing new software from scratch.

<u>The Stages of Reuse model are:</u>

- **Component Analysis:** Based on the requirements, developers search for existing components that could work. These components might only cover part of what's needed.
- **Requirements Modification:** The requirements are adjusted to fit the available components. If some parts don't fit, the team goes back to search for other options.
- **System Design with Reuse:** The overall structure of the system is created with the reusable components in mind. If something essential is missing, new parts are designed.
- **Development and Integration:** Any software not available as reusable parts is created from scratch. Then, all components and off-the-shelf systems are combined to form the final system.

d) Integration and Configuration/ Reuse oriented SE contd...

Advantages of Reuse-Oriented Software Engineering:

- **Less Development Work**: By using existing components, less new software needs to be created, which saves time and money.
- **Faster Delivery**: Since parts of the system are already made, the overall software can be delivered more quickly.

Disadvantages:

- **Compromises on Requirements**: Sometimes, existing components don't meet all the needs, leading to a system that may not fully satisfy users.
- **Less Control Over Updates**: When using reusable components, you can't control when or how they are updated, which might affect how your system works over time.

When to Use Re-Use Oriented Model:

- **Existing Components are Available:** If there are software components or systems that can be reused for your project, this model helps save time and effort.
- **Faster Development is Needed:** When you need to develop software quickly, reusing components can speed up the process.
- **Budget Constraints:** If you want to reduce development costs, using existing software can lower expenses.
- **Stable Requirements:** When your requirements are well-defined and not likely to change significantly, this model works well since it relies on available components.

Software Process Framework / Process Model:

A **process framework** is like a basic structure for managing software projects. It includes:

Framework Activities: Key activities that apply to all software projects, no matter how big or small. These activities include:

- **Communication:** Sharing information among team members.
- **Planning:** Organizing tasks and timelines.
- **Modeling:** Creating representations of the system.
- **Construction:** Building the software.
- **Deployment:** Releasing the software to users.

Umbrella Activities: Ongoing tasks that help manage and improve the entire software process, such as:

- **Project Tracking and Control:** Keeping track of progress.
- **Risk Management:** Identifying and handling potential problems.
- **Quality Assurance:** Ensuring the software meets quality standards.
- **Technical Reviews:** Evaluating the work at different stages.
- **Configuration Management:** Controlling changes to the software.
- **Reusability Management:** Managing parts of the software that can be reused in future projects.

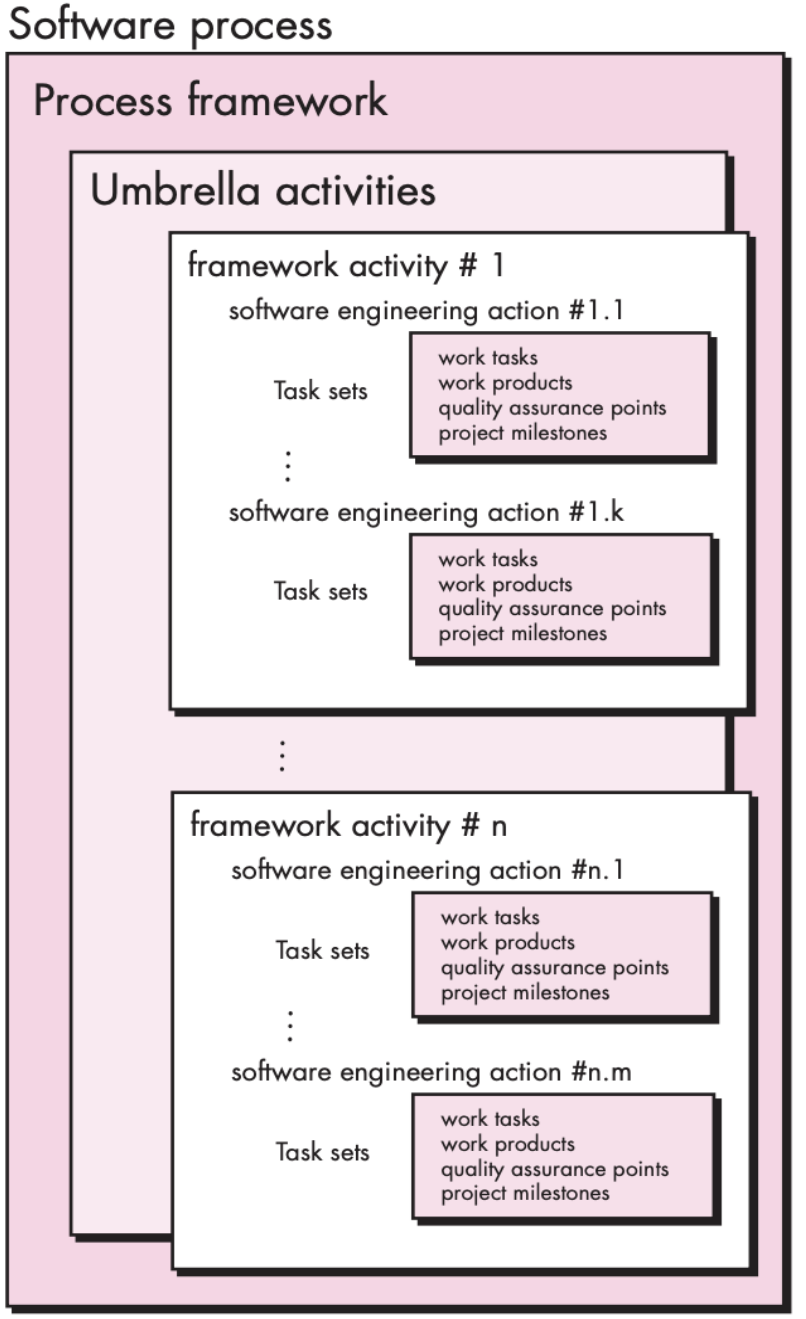
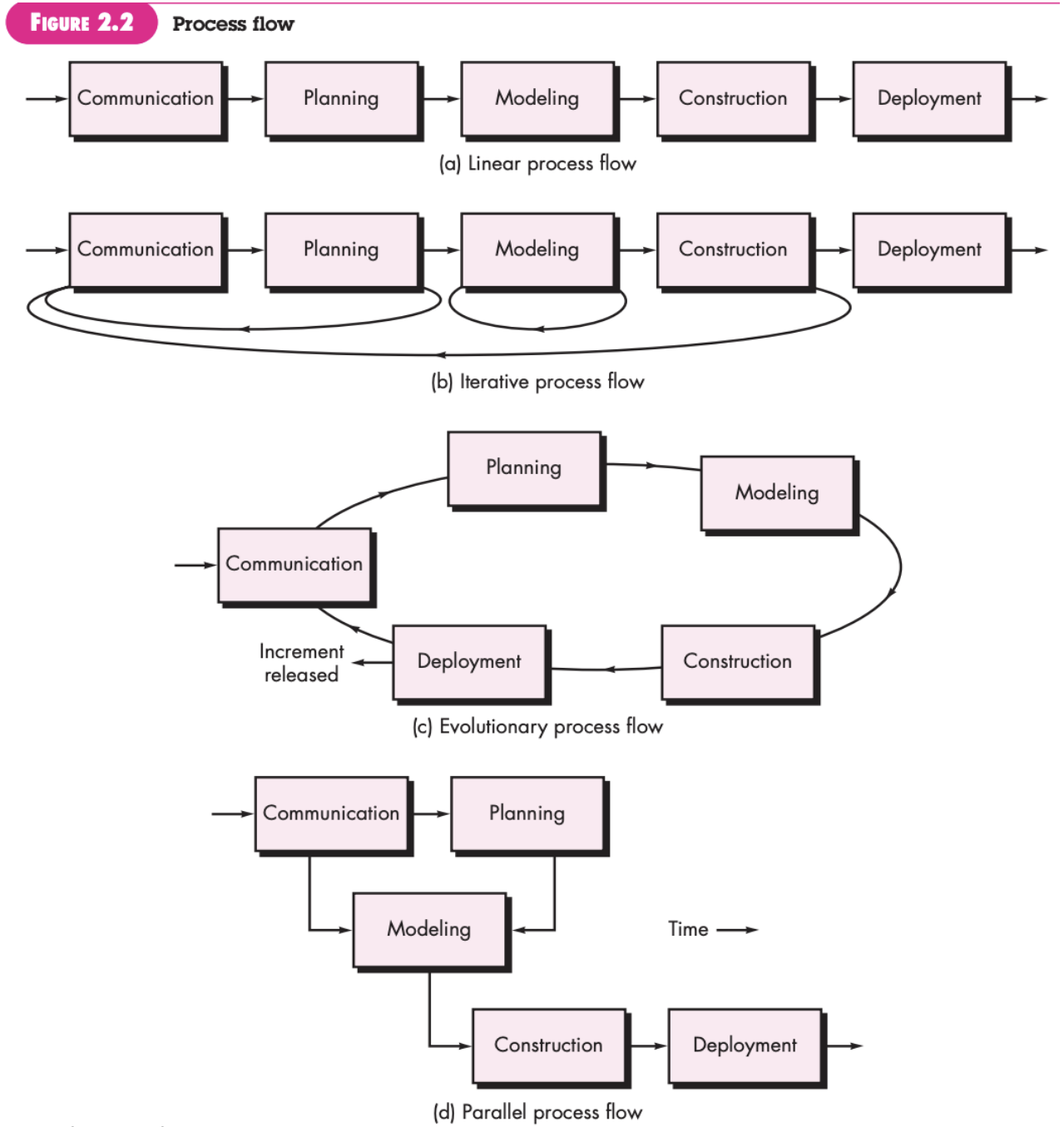


Fig: S/W process Framework

Software Process Framework / Process Model:

One important part of the software process is **process flow**, which shows how the framework activities are organized in terms of sequence and timing. Here are the different types of process flows:

1. **Linear Process Flow:** Activities are done one after the other in a straight line. You start with communication and end with deployment.
2. **Iterative Process Flow:** You repeat some activities multiple times before moving on to the next one. This allows for refining and improving parts of the software.
3. **Evolutionary Process Flow:** Activities are done in a circular way. Each time you go through the cycle, you get a more complete version of the software.
4. **Parallel Process Flow:** Some activities happen at the same time. For example, you might work on modeling one part of the software while also constructing another part.



Software Process Activities:

The four main S/W Process activities are:

a) **S/W Specification**: Defining what the software should do.
b) **S/W Design & Implementation**: Writing and building the software.
c) **Validation**: Testing to ensure the software works correctly.
d) **Evolution**: Making changes and improvements over time.

Different Approaches:

- In the **waterfall model**, these activities happen one after the other in a fixed order.
- In **incremental development**, these activities overlap and happen at the same time.

a) Software Specification (or requirements engineering) is the process of figuring out what a software system needs to do and any limitations it must follow. This step is very important because mistakes made here can cause problems later when designing and building the system.

Here's a breakdown of the process:

- **Understanding Needs**: Identify what services and features users want from the software.
- **Identifying Constraints**: Find out any restrictions on how the software should work or be developed.

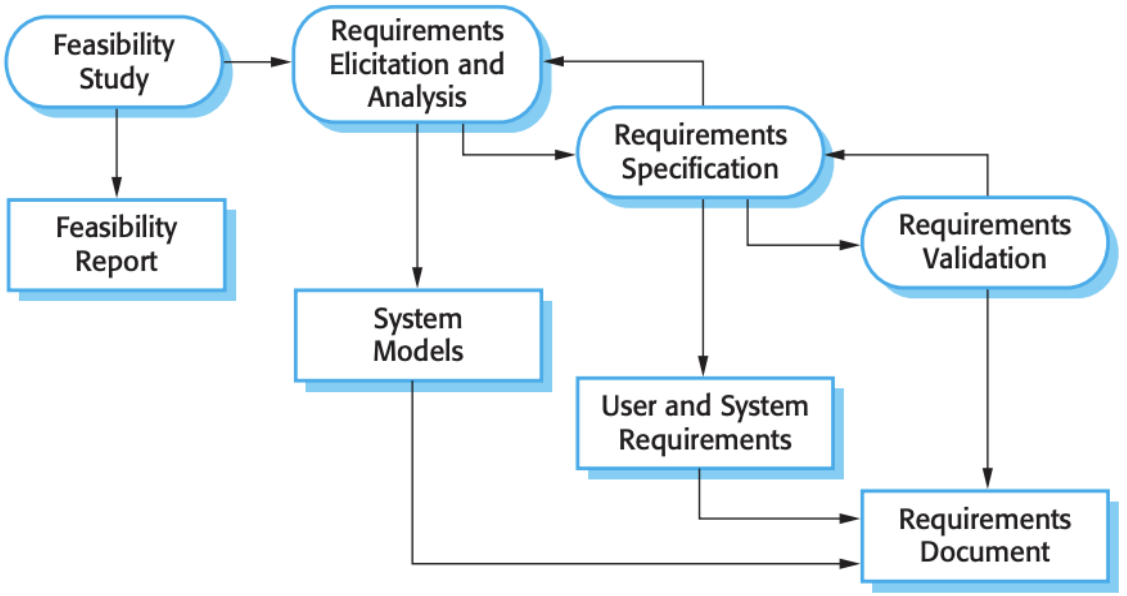


Fig: Requirement Engineering Process

The requirements engineering process involves four main activities:

i. **Feasibility Study**: This is a quick assessment to see if the user's needs can be met with current technology. It checks if the proposed system will be worth the cost and if it can be built within the budget. The outcome helps decide whether to continue with more detailed planning.

Software Process Activities:

ii. Requirements Elicitation and Analysis: In this step, you gather information about what the system should do. This involves observing existing systems, talking to users and stakeholders, and analyzing tasks. You might create models or prototypes to better understand what the system needs to achieve.

iii. Requirements Specification: Here, the information collected is turned into a clear document that defines the system's requirements. This document includes two types of requirements:

- **User Requirements**: General statements about what users expect from the system.
- **System Requirements**: Detailed descriptions of the specific features and functions the system should provide.

iv. Requirements Validation: This activity checks if the requirements are realistic, consistent, and complete. During this review, any errors or issues in the requirements document are found and need to be fixed.

- The activities in the requirements process overlap instead of happening in a strict order. While analyzing the requirements, you might find new needs. In agile methods like extreme programming, users collaborate closely with the development team to build requirements step by step, focusing on what's most important to them.

b) Software Design and Implementation:

- The implementation stage of software development involves turning a system specification into a working software system.
- This includes both software design and programming. If using an incremental approach, the software specification may also be refined during this stage.
- Software design describes how the software will be organized, covering data structures, how parts of the software will connect, and sometimes the steps (algorithms) used to solve problems.
- Designers don't create a final design right away; instead, they develop it step by step, adding details and revisiting earlier designs to make improvements.

b) Software Design and Implementation:

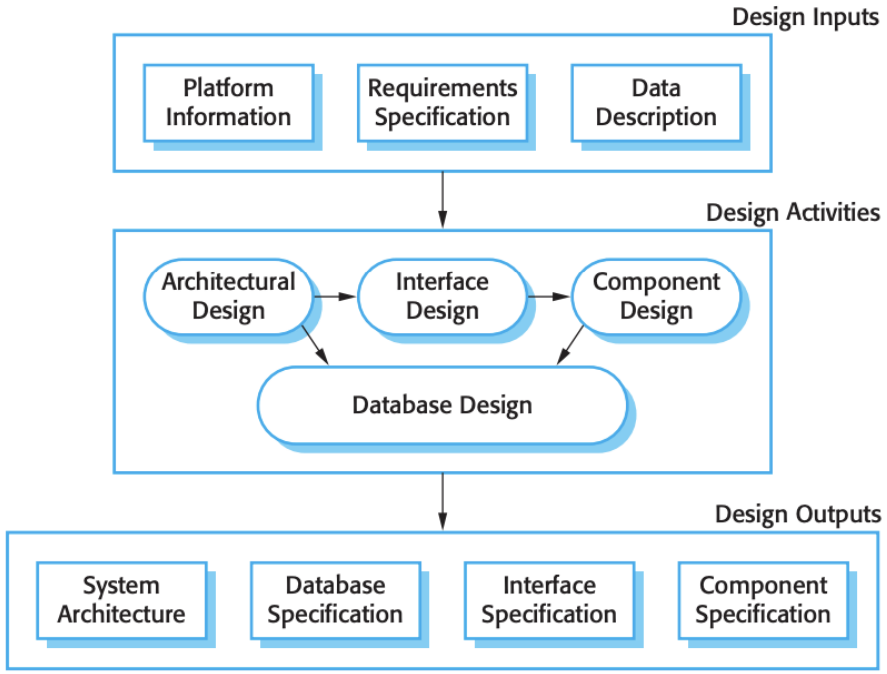


Fig: A general model for Design Process

In Above Figure: The design process includes ***Design inputs, Design activities, and Design outputs*** that guide the development of the software. For example, inputs might include user requirements, activities could involve creating diagrams, and outputs would be design documents that outline the software structure.

- The design process stages aren't strictly sequential; they often overlap, and feedback from each stage leads to reworking the design.

Figure outlines four key design activities:

  - **Architectural design:** Decide the overall system structure, key components (like modules), their connections, and how they're spread across the system.
  - **Interface design:** Define how components will interact. Clear interface details let each component work independently, so development can happen in parallel.
  - **Component design:** Plan how each part of the system will work. This could be a simple outline of its function, a list of tweaks to a reusable part, or a detailed model that can generate code.
  - **Database design:** Structure the data and decide how it will be stored in a database. This can involve using an existing database or creating a new one.

- The design process creates outputs that show how the system will work. For critical projects, these outputs are detailed documents.
- In model-driven methods, outputs often diagrams, and code can sometimes be generated from them. In agile, designs are often part of the code itself rather than separate documents. Older structured methods used visual models to plan the system, which evolved into today's model-driven development (MDD). In MDD, different levels of models are created, from general to specific, to guide development and sometimes even generate the final code.

c) Software Validation:

- Software validation, also known as verification and validation (V&V), ensures a system is made according to its specifications (Verification) and meets customer expectations (Validation).
- The main method for validation is testing, where the software runs with test data to check for issues. Validation also includes other checks, like reviews, throughout the software development process.
- For larger systems, testing is done in stages rather than all at once. First, individual components are tested (Unit Test), then the combined system (Integration Test), and finally, the entire system (System Test) with real customer data. As issues are found and fixed, parts of the testing process may need to be repeated.

Fig: Stages of Testing

The stages in the testing process are:

- **Development Testing:** Individual parts of the system, like functions or object classes, are tested separately by the developers. Automated tools like JUnit are often used to run these tests quickly, especially when changes are made to the components. Development Testing often include **Unit Testing**.
- However, development testing can also include other types of testing beyond unit testing, such as:
  - **Integration testing:** Testing interactions between multiple units to ensure they work together.
  - **Regression testing:** Running existing tests again after changes are made to ensure no new issues were introduced.
- **System Testing:** All components are combined and tested as a full system to check for errors that might arise from how the parts interact with each other. This stage ensures that both functional requirements (what the system should do) and non-functional requirements (like speed and security) are met.
- **Acceptance Testing:** This is the last step before the system goes live. The system is tested with real data provided by the customer to ensure it works as expected and meets their needs. This step may reveal issues that weren't apparent with test data, helping confirm that the system is ready for use.

c) Software Validation:

Testing phases in a plan-driven software process:

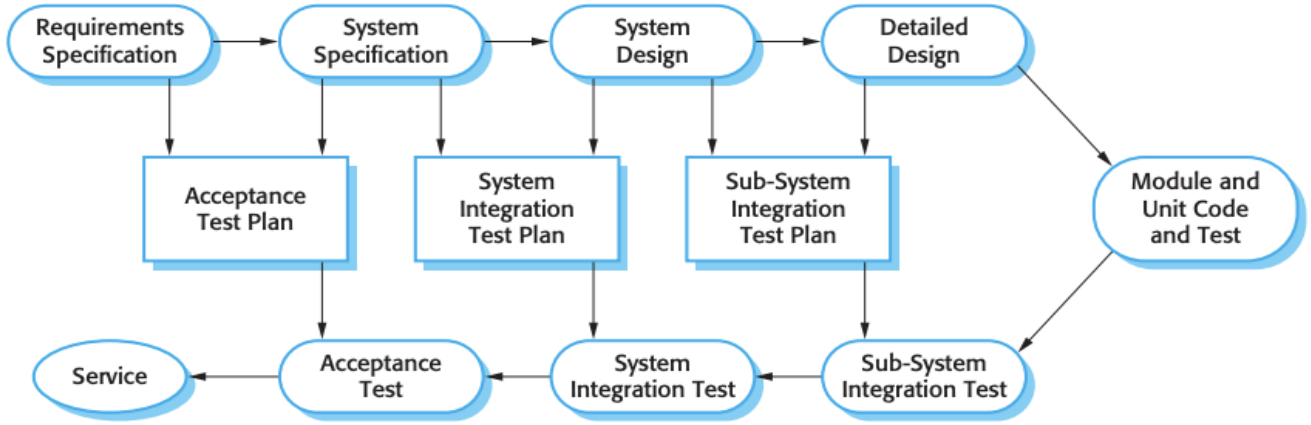


Fig: Testing phases in a plan-driven software process

In plan-driven processes, (for critical systems), testing is based on detailed test plans made from the system's requirements and design. An independent team follows these plans to check the system. This approach is often visualized as the **V-model**, where development and testing activities are linked.

- **Acceptance testing** (or **alpha testing**): Used for custom systems with one client, testing continues until both the developer and client agree the system meets their needs.
- **Beta testing**: Used for products intended for a wider market, where the system is given to selected users. They use it and report any issues, helping to find real-world errors. The system is then improved before a general release.

d) Software Evolution:

- Software evolution is about changing and updating software after it's built. Software is flexible, so it's easier and cheaper to modify than hardware.
- Traditionally, people viewed software development and software maintenance as separate activities, with development seen as creative and maintenance viewed as dull and uninteresting.
- But now, we understand that they're interconnected. Most software isn't brand new; it changes over time to meet new needs.
- So, instead of seeing them as two different things, it's better to think of software development as a continuous process where software is regularly improved and updated as shown in figure.

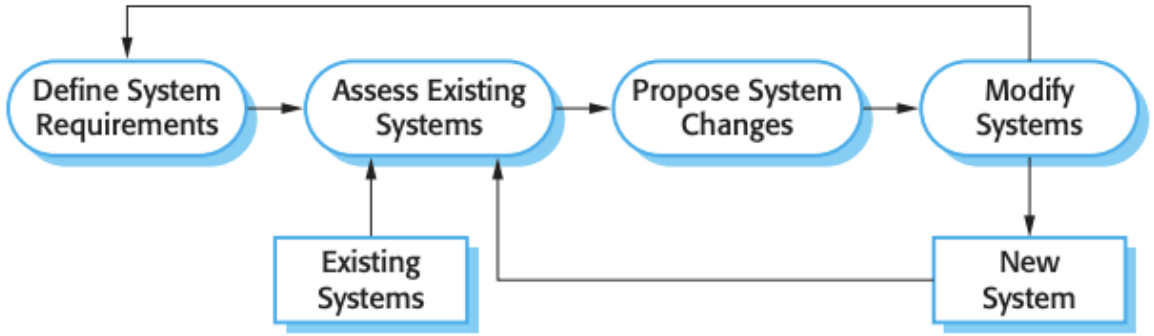


Fig: System Evolution

Coping with Change:

- Change happens often in large software projects as businesses shift priorities or new technology becomes available. New technologies can also bring new ways to design and build the system.
- No matter which software development model is used, it's important that it can handle these changes. However, changes can increase development costs because they often require revisiting work that's already been done. This extra work is called rework.
- For example, if new requirements come up after analyzing existing ones, the requirements analysis process has to start over. This might lead to redesigning the system, updating programs that have already been created, and retesting the system to ensure everything works correctly.

To reduce rework costs, two main strategies can be used:

1. **Change Avoidance**: This involves planning for possible changes early. For instance, a prototype can be created for customers to test key features and adjust their requirements before full development starts.
2. **Change Tolerance**: This approach makes it easier to handle changes later on, often through incremental development. New changes can be added in future increments or by modifying only a small part of the system if needed.

Three methods help manage changing requirements / coping with change:

- **System Prototyping, Incremental Delivery, Boehm's Spiral Model**

a) Prototyping:

A prototype is an early version of a software system, created to test ideas, explore design options, and better understand the problem. Rapidly creating and updating the prototype keeps costs down and allows stakeholders to try it out early in the process.

Prototyping helps by:

- **Requirements Gathering:** Clarifying and confirming system requirements.
- **System Design:** Testing software solutions and refining the user interface.

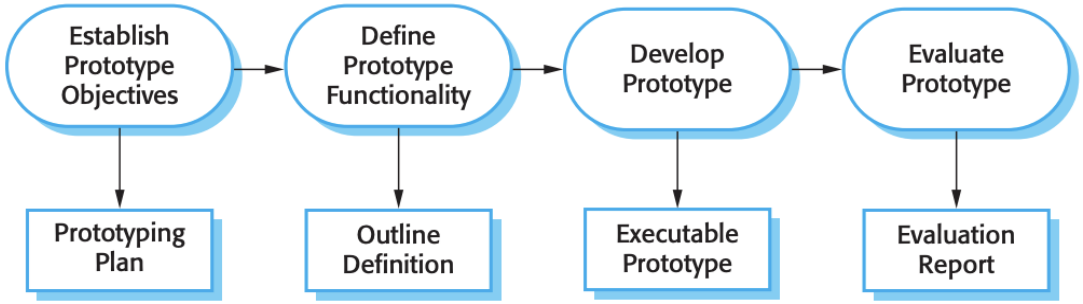


Fig: The process of prototype development

Coping with Change:

a) Prototyping:

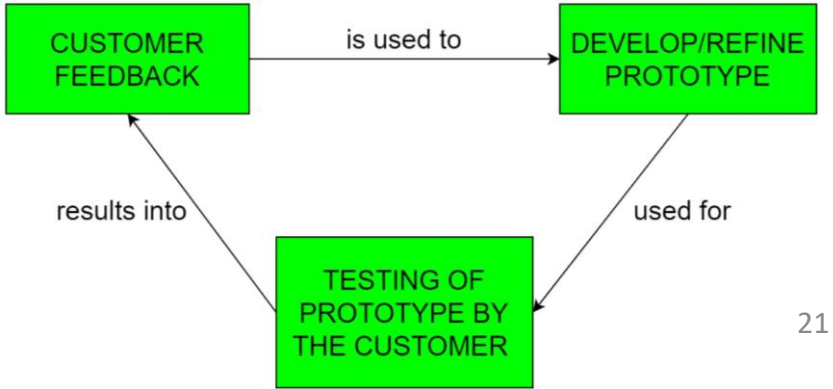
The Phases are:

- **Establish Prototype Objectives:** In prototype development, it's essential to clearly define the purpose of the prototype from the beginning. Goals might include testing the user interface, checking if the system's functions work as intended, or showing the project's feasibility to management. A single prototype can't achieve all these goals, so if the objectives are unclear, users or managers may misunderstand its purpose and miss its benefits.
- **Define Prototype Functionality:** Focus on the most important features. To save time and costs, some functions, like error handling or high reliability, might be left out unless they are essential for the prototype's purpose.
- **Develop Prototype:** Build a prototype to test specific features or designs of the software. This allows users to provide feedback and refine requirements early in the development process.
- **Prototype Evaluation:** Create a plan for users to test the prototype according to its goals. Give them time to learn how to use it so they can spot any mistakes or missing features.

Problem with prototype model: A problem with prototypes is that users might not use the prototype the same way as the final system. The client testers may not be regular users, and if they don't get enough training or if the prototype is slow, they might skip some features. When they try the faster final system, they might act differently.

When to use Prototyping Model?

- Whenever the customer is unknown about the exact requirements (input, processing, output) at that time prototyping model is preferred.
- In this model, the prototype of end product is developed at first and then tested and refined using customer feedback. Customer feedback is taken until and unless customer is satisfied with the final acceptable prototype.
- In this prototyping model, the system is partially implemented before or during the Analysis phase so that customer can see the product in early lifecycle.



Coping with Change:

a) Prototyping:

How prototyping process starts?

- Interviewing the customer and developing incomplete high-level paper model.
- Building initial prototype using same incomplete high-level paper model. Initial prototype provides software's basic functionality.
- Whenever the customer figures out the problems, prototype is refined and problems are eliminated.
- Process contineus until user accepts the final product.

- Prototypes do not have to be executable to be useful. Paper mock-ups of the user interface can effectively help users improve the design and test how they would use the system. These are cheap to make and can be ready in a few days.
- Another type is called a Wizard of Oz prototype, where users interact with a user interface, but a person behind the scenes handles their requests and gives the right responses.

Advantages of Prototyping model:

- User is actively involved in the development phase so very less chance of project fail.
- Quicker user feedback available which helps for better solution.
- Errors can be detected much earlier.
- Easy to identify missing functionalities.

Dis- advantages of Prototyping model:

- User has the expectation of having same performance of final system and prototype.
- poor documentation because of continuously changing customer requirements.
- Confusion between final product and prototype.
- Customers sometimes demand the actual product to be delivered soon after seeing an early prototype.

Coping with Change:

Types of Prototyping Model:

i. Rapid (Throwaway) Prototyping:

- Rapid prototyping is a quick and easy way to create a model of a system. It allows designers and users to try out different ideas and get feedback fast.
- It's called "throwaway prototyping" because the prototype is only used for a short time, like one sprint in Agile development. After getting feedback from all stakeholders and making changes, the prototype serves as a guide/ reference for the final design. Once the sprint is over, the prototype is discarded, and a new one is made for the next phase.
- Even slower prototypes, like paper mock-ups, can be considered "throwaway" since they are not kept after the development cycle.

Sprint → A sprint is a set period during which specific work must be completed and made ready for review in Agile development, particularly in frameworks like Scrum.

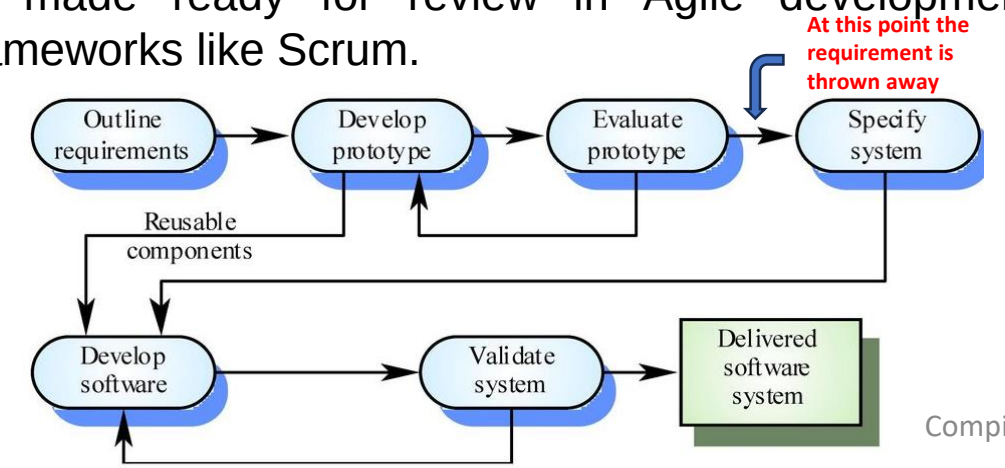
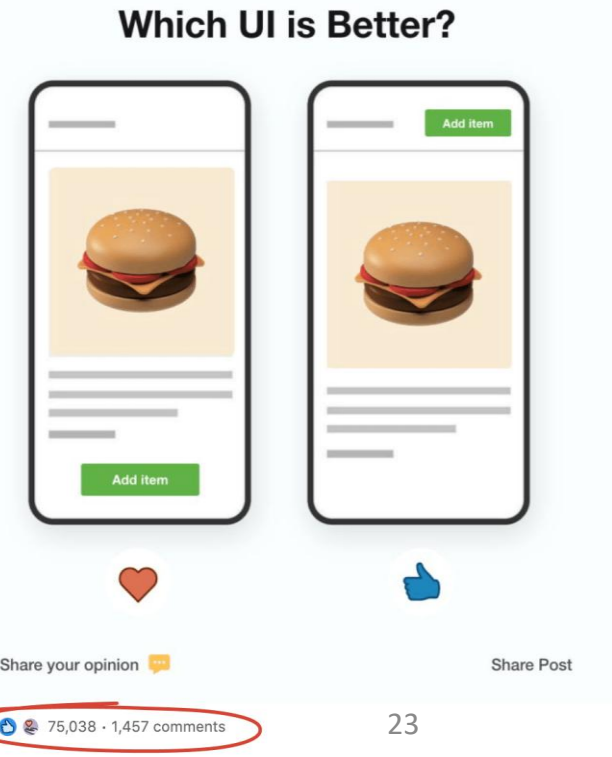


Fig: Throwaway Prototyping

For example,

- *Let's say a software development team is unsure about the best way to design the user interface for a new mobile app.*
- *They might create a quick, throwaway prototype to explore different interface layouts and gather feedback from users.*
- *Once they have enough information to make a decision, they discard the prototype and use the insights gained to finalize the design requirements for the app.*



Coping with Change:

Types of Prototyping Model:

i. Rapid (Throwaway) Prototyping:

Managers sometimes pressure developers to turn throwaway prototypes into final products, especially when the project is delayed. However, this usually causes issues:

- The prototype may lack essential qualities like performance, security, and reliability.
- Prototypes change quickly, often without documentation, making them hard to maintain long-term.
- Quick changes can weaken the system's structure, leading to costly maintenance.
- Prototypes often don't meet quality standards since they were meant for testing, not final use.

Coping with Change:

Types of Prototyping Model:

ii. Evolutionary Prototyping:

- In evolutionary prototyping, you begin by modeling parts of the target system and, if the prototyping process is successful, you evolve the rest of the system from those parts (McConnell, 1996).

Eg: Let's say a company wants to develop a new online shopping platform. In evolutionary prototyping, they would start by modeling and developing a few key features or modules of the platform, such as:

- *User Registration, Product Listing, Shopping Cart etc.*

Once these initial modules are developed and tested, then continue to evolve and expand the system by adding more features

- *Payment Gateway Integration, User Reviews and Ratings etc.*

- An evolutionary prototyping life-cycle model shows the iterative process of refining the prototype until it's ready to release.(Figure 6-8).
- A key point of this approach is that the prototype becomes the final production system. This means you often begin with the most difficult and uncertain parts of the system.

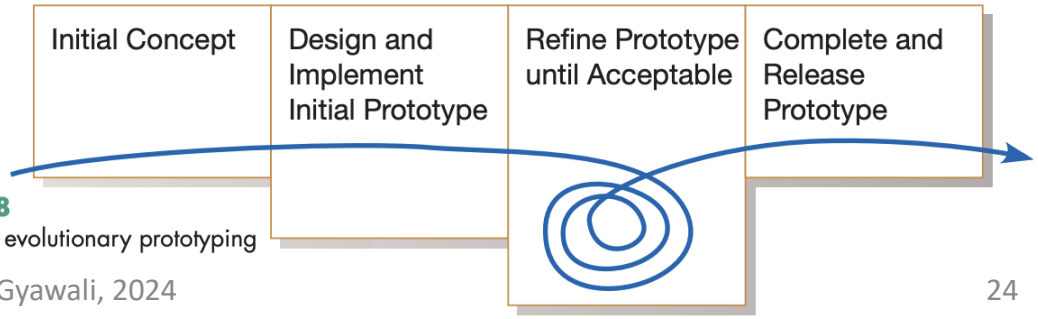


FIGURE 6-8
McConnell's evolutionary prototyping model

Coping with Change:

Types of Prototyping Model:

Difference between Evolutionary Vs Throwaway Prototyping

Aspect	Evolutionary Prototyping	Throwaway Prototyping
Purpose	To gradually refine and improve the prototype.	To gather feedback quickly before building the final system.
Development	The prototype is continuously developed and modified.	The prototype is created quickly and is not refined.
Final Product	The prototype evolves into the final product.	The prototype is discarded after feedback is gathered.
User Feedback	Feedback is collected throughout the development process.	Feedback is gathered after the prototype is built.
Cost of Changes	Changes are often less costly since the prototype is improved over time.	Changes can be costly since the prototype is not built for longevity.

b) Incremental Delivery:

- Incremental delivery is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in an operational environment.
- In an incremental delivery process, customers identify, list out, and prioritize the services to be provided by the system. They identify which of the services are most important and which are least important to them.
- The system is then divided into increments, each offering some of the functionality. The highest-priority features are developed and delivered first, so customers can start using parts of the system sooner.
- After the system parts (increments) are chosen, detailed requirements for the first part are defined, and that part is developed. While working on it, requirements for future parts can be analyzed, but changes for the current part aren't allowed.
- Once a first part is finished, it's delivered to the customer to use. This lets customers try out part of the system early, helping them better understand and clarify needs for future parts. Each new part adds to the existing system, increasing its functionality.

Coping with Change:

b) Incremental Delivery:

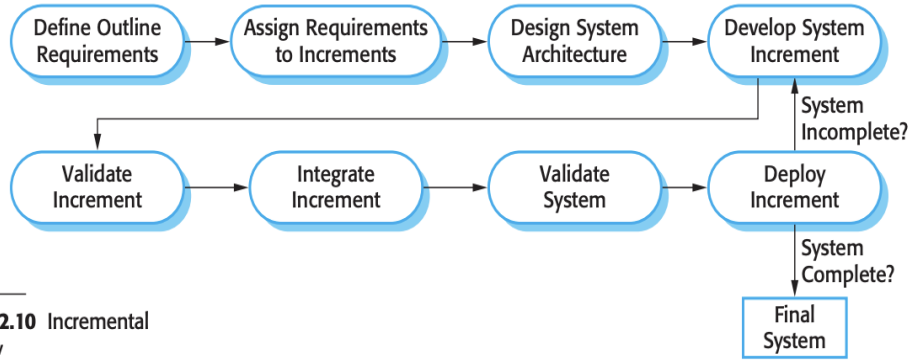


Figure 2.10 Incremental delivery

Advantages of Incremental Delivery:

- Customers can try out early versions of the system and adjust their needs for later parts. Since these are real parts of the system, they won't have to relearn them.
- Customers can start using the software sooner, as the first part delivered meets their most critical needs.
- It's easy to make changes as the system is built.
- Important services get delivered and tested first, making them more reliable and less prone to failures when used.

Problems with Incremental Delivery:

- It's hard to find common features needed for all parts without a full plan from the start.
- If the new system replaces an old one, users may resist using an incomplete system, making it hard to gather useful feedback.
- Incremental development doesn't align with the traditional contract model, where a full system specification is needed from the start. This can be challenging for large clients, like government agencies, that require a full contract up front.

When Incremental Delivery doesn't suit?

For some systems, incremental development and delivery may not work well. This includes:

a) very large systems with teams spread across different locations,→ Diff. Time Zone, Language Barrier in communication

b) embedded systems that rely on hardware development,

c) critical systems where all requirements need careful analysis for safety or security reasons.

These all systems still face changing requirements. To handle these challenges and still get some benefits of incremental development, a prototype can be created and improved step-by-step. This prototype serves as a way to test and explore system requirements and design options.

Coping with Change:

c) Boehm's spiral model:

- Spiral model combines the features of both waterfall model and prototyping model.
  - Feature of waterfall model → goes through different steps, like planning, designing, building, and testing.
  - Feature of prototyping model → allowing for iterative development and refinement.
- Spiral model is similar to the incremental model but in spiral model there is more focus on Risk Analysis.
  - Risk: Uncertainty that might affect a project, activity, or situation.
  - Risk Analysis: At the start of each spiral, the team looks for potential risks like: technology issues, budget concerns, or changes in what the customer wants. Once risks are identified, the team analyzes them to figure out how likely they are and what impact they could have on the project.
- The concept of spiral model was introduced in late 1980s by Barry Boehm who incorporated the idea of "Risk Analysis" in a Software Development Life Cycle Model.
- Multiple spirals indicates the iteration of software product along with cost and risk analysis.

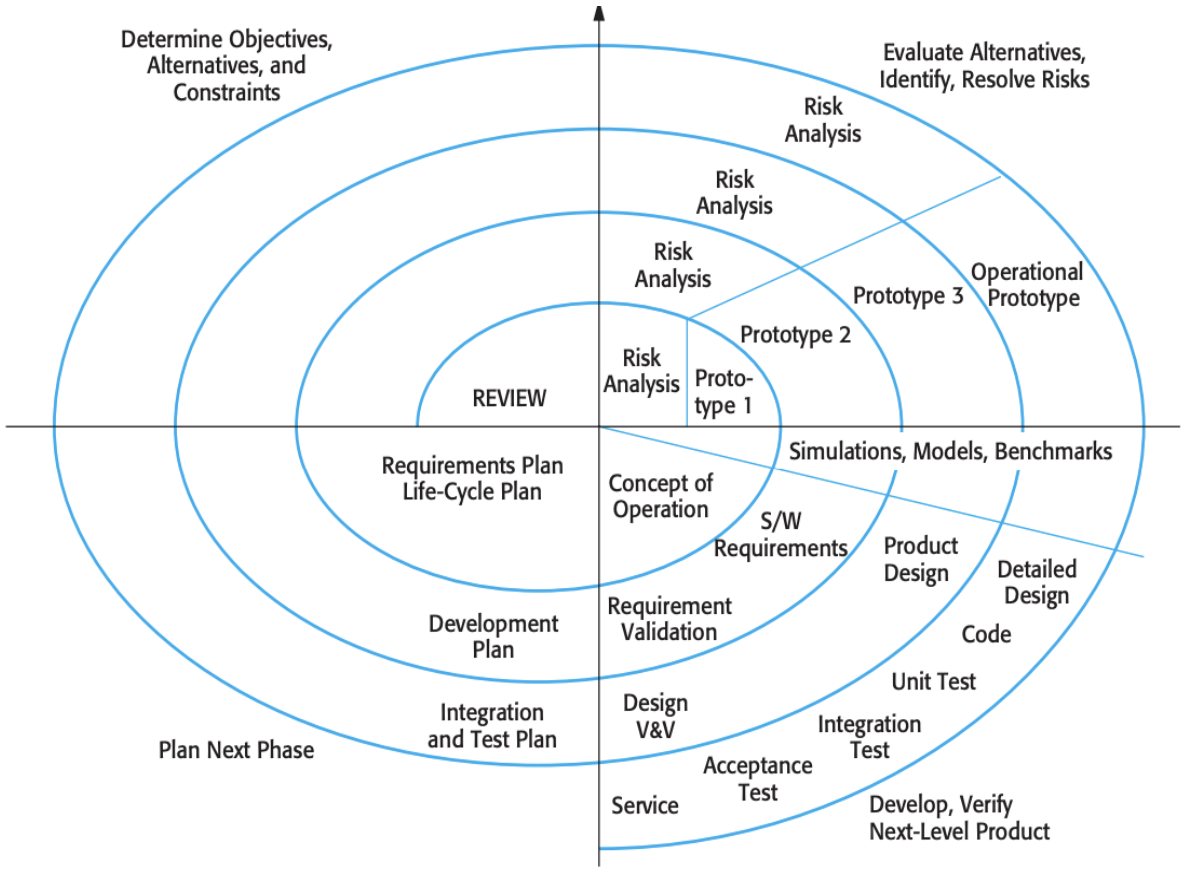


Fig: Boehm's Spiral Model for a Software Process

Coping with Change:

c) Boehm's spiral model:

Each loop in the spiral is split into four section:

1. **Objective Setting:** In this phase, clear goals are set for the project. Limitations for the process and final product (Budget constraint, Deadline, Resource Availability, Technical Constraints) are noted, and a detailed management plan is created. Risks are identified, and backup plans may be made based on these risks.

2. **Risk Assessment and Reduction:** For each risk in the project, we analyze it in detail and take steps to reduce it.
    - Risk is an Expectation of Loss that may or may not occur in the future.
    - The possible risk that can come while developing the software product are listed and their possible solution are generated in this phase.
    - Risk like → known risk (unrealistic delivery of product), Technical risk (Threatens s/w quality and timelines), Predictable Risk, Unpredictable Risk.

For instance, if there's a chance that the requirements might not be suitable, we can create a prototype system to help clarify and improve those requirements.

3. **Development and Validation:** After assessing risks, we choose a suitable development model for the system.

- For example, if user interface issues are a big concern, we might use throwaway prototyping.
- If safety is the main focus, we could use a formal transformation approach.
    - A formal transformation approach involves using mathematical methods and specifications to transform a system's requirements into a working software system. **Like: Z, VDM, B languages.**
- If the biggest risk is integrating different parts of the system, the waterfall model might be the best option.

4. **Planning for Next Iteration:** The project is reviewed/ feedback provided by customer and a decision made whether to continue with a further loop of the spiral.

- If the decision is to proceed, detailed plans are created for the next phase of the project.

Coping with Change:

c) Boehm's spiral model:

Why Spiral Different than other Models ?

- The spiral model is different from other software development models because it focuses on managing risks.
- Each cycle starts by setting clear goals, like how the software should perform and what features it should have.
- Then, different ways to meet those goals are explored, and potential challenges are identified.
- After figuring out the risks, some development work is done, and plans are made for the next phase.

Advantages of Spiral Model:

- Highly Customizable : Can be used when changes are frequent.
- Customer can see product in early lifecycle.
- Risk analysis feature/ability.
- Lesser chances of project failure.
- Suitable for larger project with higher risk.
- Good for large and complex project.

Dis- advantages of Spiral Model:

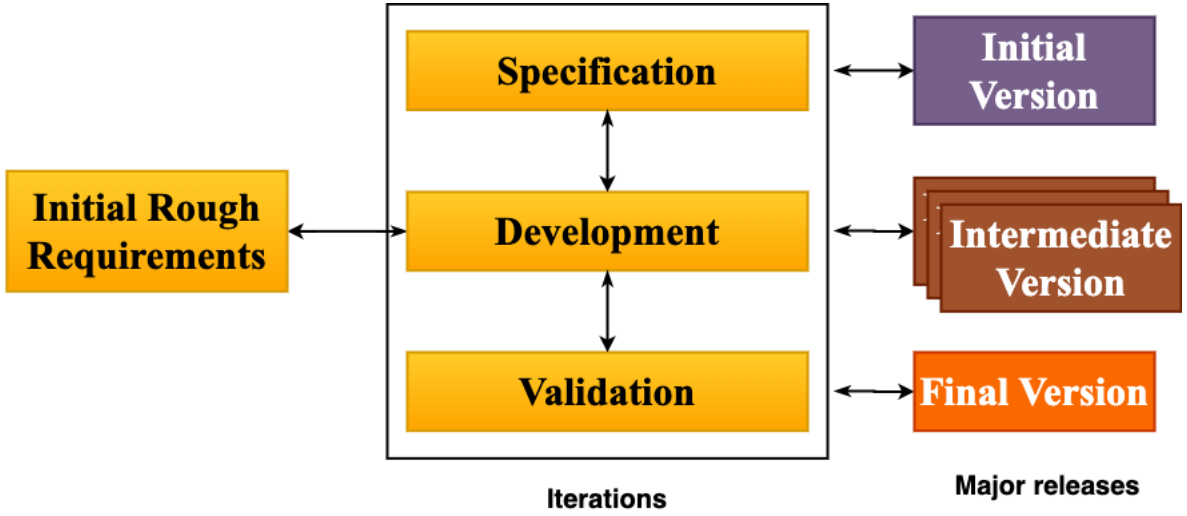
- Complex software development model.
- Not advisable to use with smaller scale projects.
- Highly experienced risk analysts are required for making risk analysis.
- Not suitable on fixed budget development: Cost revised in each iteration.

When to use spiral model?

- **High Risk**: Your project has a lot of unknowns or risks, like new technologies or unclear requirements.
- **Large Projects**: The project is complex and requires careful planning and frequent reassessment.
- **User Feedback**: You want regular input from users to improve the system over time.
- **Changing Requirements**: The project may need changes during development, and you want to adapt easily.
- **Budget Flexibility**: You can afford to invest in multiple development cycles to refine the product.

Evolutionary Model/ Development:

- Core modules of the software are developed at first. And the same core modules are refined called iterations.
- New functionalities are added at the iteration and existing functionalities can be changed according to user's requirements or feedback.
- Each iteration is developed using mini- waterfall model, and at the end of each iteration tested, integrated executable system is developed and finally deployed on the customer's site. This is also called a minor release of the software product.



- At the end of development, existing features might be improved, or some might be slightly changed. Sometimes, new features are also added. This process is called an "increment," making it a type of incremental development. This final, big update is known as a major release of the software product. Before a major release, there are usually several minor releases that make smaller changes or improvements.

Advantages:

- Users get a chance to experiment with a partially developed system much before the full working version is released.
- It is best suited to find the user requirements, the exact user requirements, and once it is delivered to the customer, the software meets the customer requirements.
- Core modules get tested thoroughly, which reduces the chances of errors in the final delivery software.
- Better management of changing requirements.

Dis-Advantages:

- The process is intangible – no regular, well-defined deliverables.
- Late to deliver a complete system because of change request from user.
- Systems may not even converge to a final version.
- Not suitable for smaller projects, can be costly, High skilled resources required for risk analysis.

Type of Evolutionary Model

a) Rapid Application Development (RAD):

- RAD model is based on Prototyping and Iterative Model in which there is less or no specific planning.
- RAD model gives less emphasis on task planning and more emphasis on Development.
- Gathers customer's requirement through workshops or focus group. Customer can test the prototype in early stage through iterative concept.
- Existing prototypes can be reused, continues integration and rapid delivery is possible.

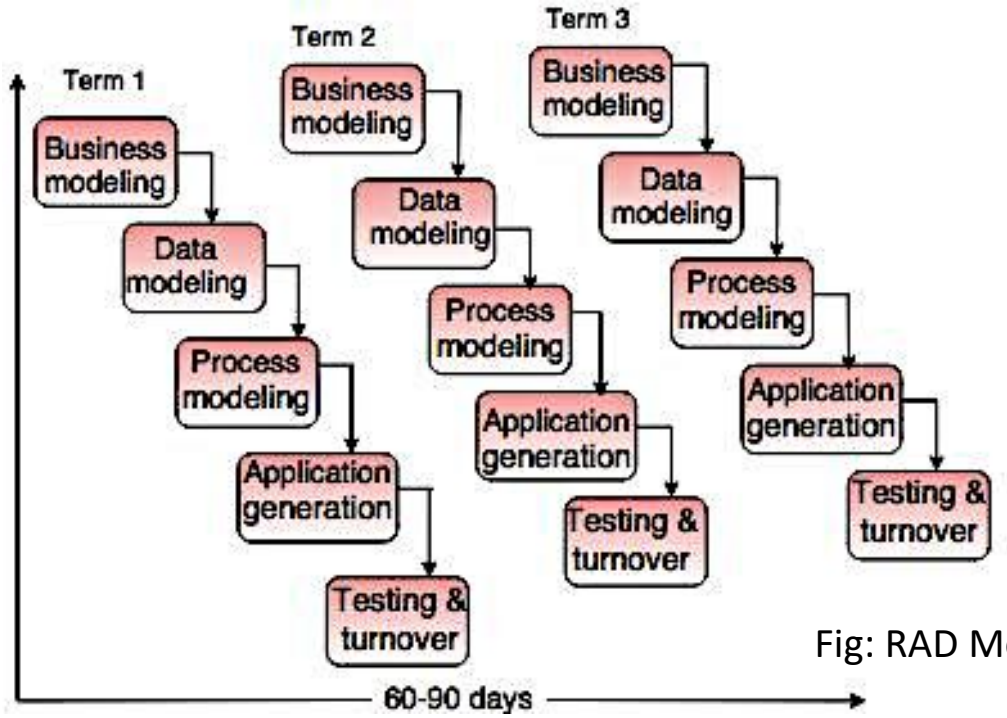


Fig: RAD Model

Stages of RAD:

- **Business Modeling:** Various meetings is held between client and requirement planning team and identifies business function and product scope.
- **Data Modeling:** Refining informations collected from Business Modeling phase into Data Objects (entities).
- **Process Modeling:** Data Objects (entities) collected in Data Modeling phase is transformed into Useful information.
- **Application Generation:** Using Different Automated CASE Tools to develop actual Prototype.
  - Case Tools → CASE tools are set of software application programs, which are used to automate SDLC activities. Eg: Rational Rose, MS-Visio
- **Testing and Turnover:** Testing all the modules and interfaces of Prototype.

When to use RAD ?

- When a system needs to be produced in a short span of time (2-3 months).
- When the requirements are known.
- When the user will be involved all through the lifecycle.
- When technical risk is less.

Type of Evolutionary Model

a) Rapid Application Development (RAD):

Advantages:

- Reduced Project Cycle Time because Reusable components are used.
- Can get User feedback at initial stages.
- Cost Reduced- Requires only fewer Developers
- Project progress and Development can be measured.

Dis-Advantages:

- Only system that can be modularized can be developed.
- RAD project can fail if developers are not able to provide a software before a deadline.
- For Smaller project RAD cannot be used.
- Requires Highly skilled professionals to use Automated CASE Tools.

Types Of Evolutionary Process Model:

a) Rapid Application Development b) Prototyping c) Spiral Model
d) Concurrent Models → Assignment

Specialized Process Models:

1) Component Based Development (CBD) / Component Based Software Engineering (CBSE):

Component → Independent S/W unit

- Component-Based Software Engineering (CBSE) follows the "Buy, Don't Build" philosophy, meaning it's often better to use existing software components rather than creating new ones from scratch.
- This approach is different from the traditional waterfall model because it focuses on assembling software from pre-made components instead of developing everything in a linear sequence.
- Essentially, CBSE aims to speed up the development process by reusing components that are already available.
- In software engineering, reuse has always been part of development, but the approach was informal in the early days. As systems have grown more complex and development timelines have shortened, reuse now requires a more organized strategy.
- (CBSE) focuses on designing and building systems by reusing software "components"—self-contained pieces of code that can be combined to create larger applications.

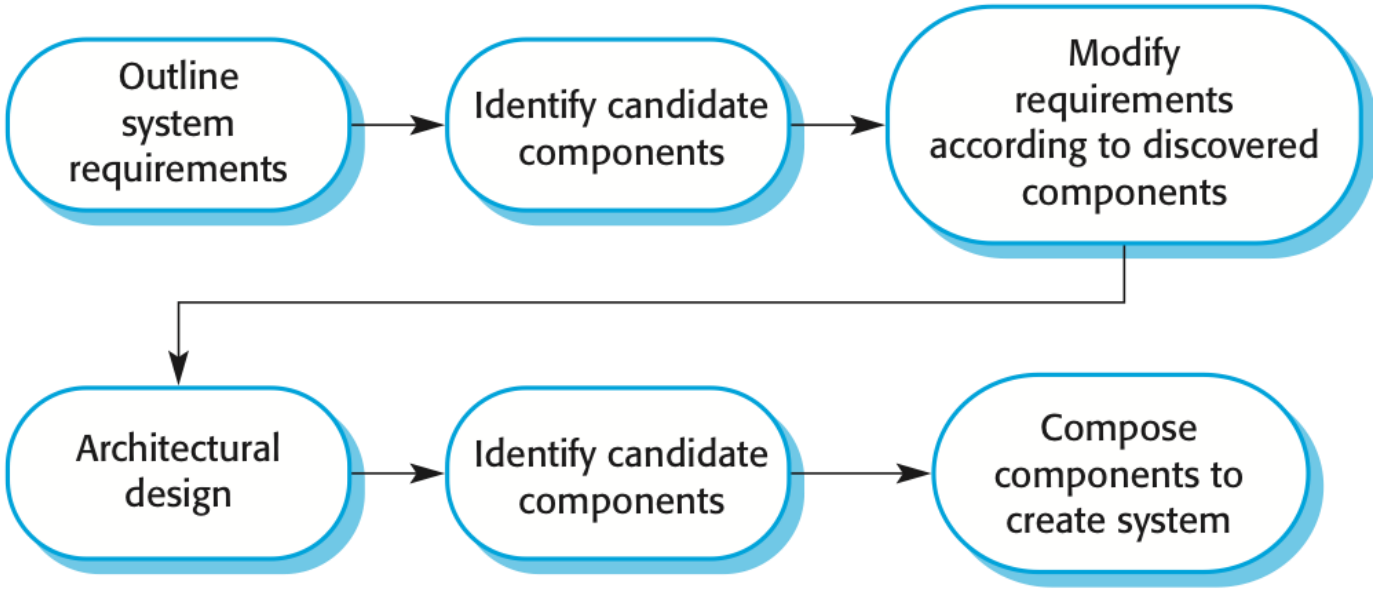
1) Component Based Development (CBD) / Component Based Software Engineering (CBSE)

- This CBSE allows developers to focus more on integration than implementation.
- However, several questions come up, such as:
  - Can we build complex systems by assembling reusable components?
  - Can this be done cost-effectively and quickly?
  - Are there incentives for engineers to reuse instead of creating new components?
  - Will management support the initial costs to build reusable components?
  - Can we create a component library that's accessible and searchable?

The Answer to the all Question is **YES !!**

- Component-Based Development (CBD) is a software approach that uses pre-made software components (often called COTS, or "commercial off-the-shelf" components) provided by vendors.
- These components have specific functions and clear interfaces, making it easier to integrate them into new software projects.
- CBD shares features with the spiral model, as it also uses an evolutionary, iterative approach to build software, but it focuses on assembling applications from existing components.

CBSE Process:



1) Component Based Development (CBD) / Component Based Software Engineering (CBSE)

CBSE Process:

- **Outline System Requirements:** Start by defining what the system needs to do. This outlines the basic functionalities and requirements for the software.
- **Identify Candidate Components:** Look for existing software components that could fulfill the outlined requirements. These components are pre-built and can save time in development.
- **Modify Requirements According to Discovered Components:** After identifying candidate components, you may need to adjust the system requirements to fit the capabilities of these components. This ensures compatibility and optimal use of resources.
- **Architectural Design:** Plan how the components will fit together in the overall system. This involves creating a structure that allows for seamless integration of the selected components.
- **Identify Candidate Components (again):** Revisit the process of finding suitable components, possibly based on the new architectural design. This may lead to discovering additional components that fit better.
- **Compose Components to Create System:** Finally, assemble the identified components into a working system, ensuring they work together as intended.

Advantages of CBSE:

- Components can be reused across different applications, saving time and effort in development.
- The system is divided into smaller, manageable components, making it easier to understand and maintain.
- Developers can focus on integrating components rather than building everything from scratch, leading to faster development cycles.
- Components can be easily replaced or upgraded without affecting the entire system.

Dis-Advantages of CBSE:

- Combining different components may lead to compatibility issues and increased complexity.
- Components may rely on specific versions of other components or libraries, complicating updates and maintenance.
- The use of multiple components can introduce overhead, potentially impacting system performance.
- Relying on third-party components can limit the control over certain aspects of the software, such as security and functionality.

CBSE vs Traditional SE → Assignment !!

Comparision of All Process Models:

Model	Well-Defined Requirements	Domain Knowledge of Team Members	Expertise of User in Problem Domain	Availability of Reusable Components	User Involvement in All SDLC Phases	Systems Complexity	Risk Analysis	Cost	Project Completion Time
Waterfall	High	Medium	Low	Low	Low	Low to Medium	Low	Low to Medium	Longer
V-Model	High	Medium	Low	Low	Low	Low to Medium	Medium	Low to Medium	Longer
Spiral	Low to Medium	High	Medium to High	Medium	High	High	High	Medium to High	Flexible
Incremental	Medium to High	Medium	Medium	High	Medium to High	Medium	Medium	Medium	Shorter
Prototyping	Low	Medium	High	Medium	High	Medium	Medium	Medium	Shorter
RAD	Low	Low to Medium	High	High	High	Medium	Medium	Medium to High	Very Short

Rational Unified Process:

- **The Rational Unified Process (RUP)** is a modern software development process model, built on concepts from the **Unified Modeling Language (UML)** and **Unified Software Development Process**.
- It's a hybrid model that combines features of various traditional process models (Waterfall, Incremental, Re-use Oriented) and supports good practices like prototyping and incremental delivery.

Three Perspectives of RUP:

- **Dynamic perspective:** Shows how the project moves through different phases over time. ***Different phases*** → Inception, Elaboration, Construction, Transition
- **Static perspective:** Describes the main activities done in the process. ***Main activities*** → Requirement Analysis, Design, Implementation, Testing
- **Practice perspective:** Provides best practices to follow for effective development. ***Best Practices*** → Iteration Development, Risk Management, Continuos Testing

RUP has four main phases, but unlike the Waterfall model, these phases focus more on business goals (Launch product quickly, lower dev. Cost, user satisfaction) than on technical tasks (Analysis, Design, Coding, Testing).

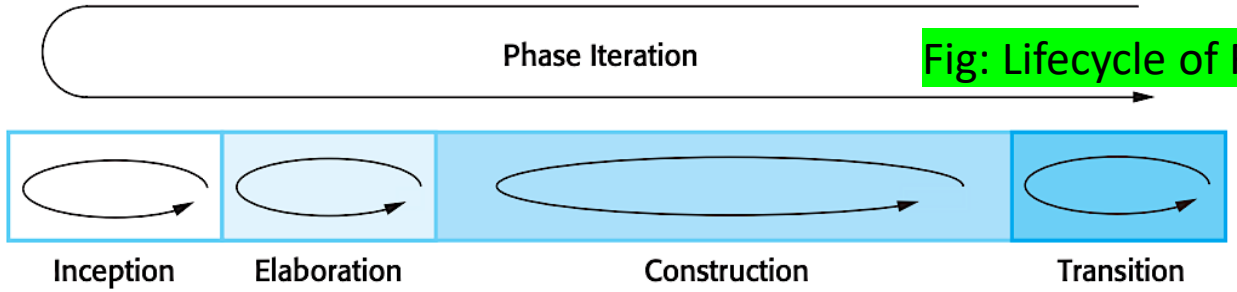


Fig: Lifecycle of RUP

- **Inception**: The goal here is to decide if the project is worth pursuing. This involves identifying all people and systems that will interact with the software and understanding how they will use it. If the project doesn't seem valuable to the business, it may be stopped after this phase.
- **Elaboration**: In this phase, the goal is to fully understand what the software needs to do, design the basic structure, create a project plan, and identify major risks. By the end of this phase, you should have a clear picture of the software's requirements, a design framework, and a development plan.
- **Construction**: This phase is about building the software. The team designs, codes, and tests different parts of the system, often working on multiple parts at the same time. By the end, you should have a functional version of the software along with documentation.
- **Transition**: The final phase focuses on moving the software from development into actual use. This involves setting it up in a real environment and resolving any issues that come up. When finished, you should have a fully operational, well-documented system for the users.

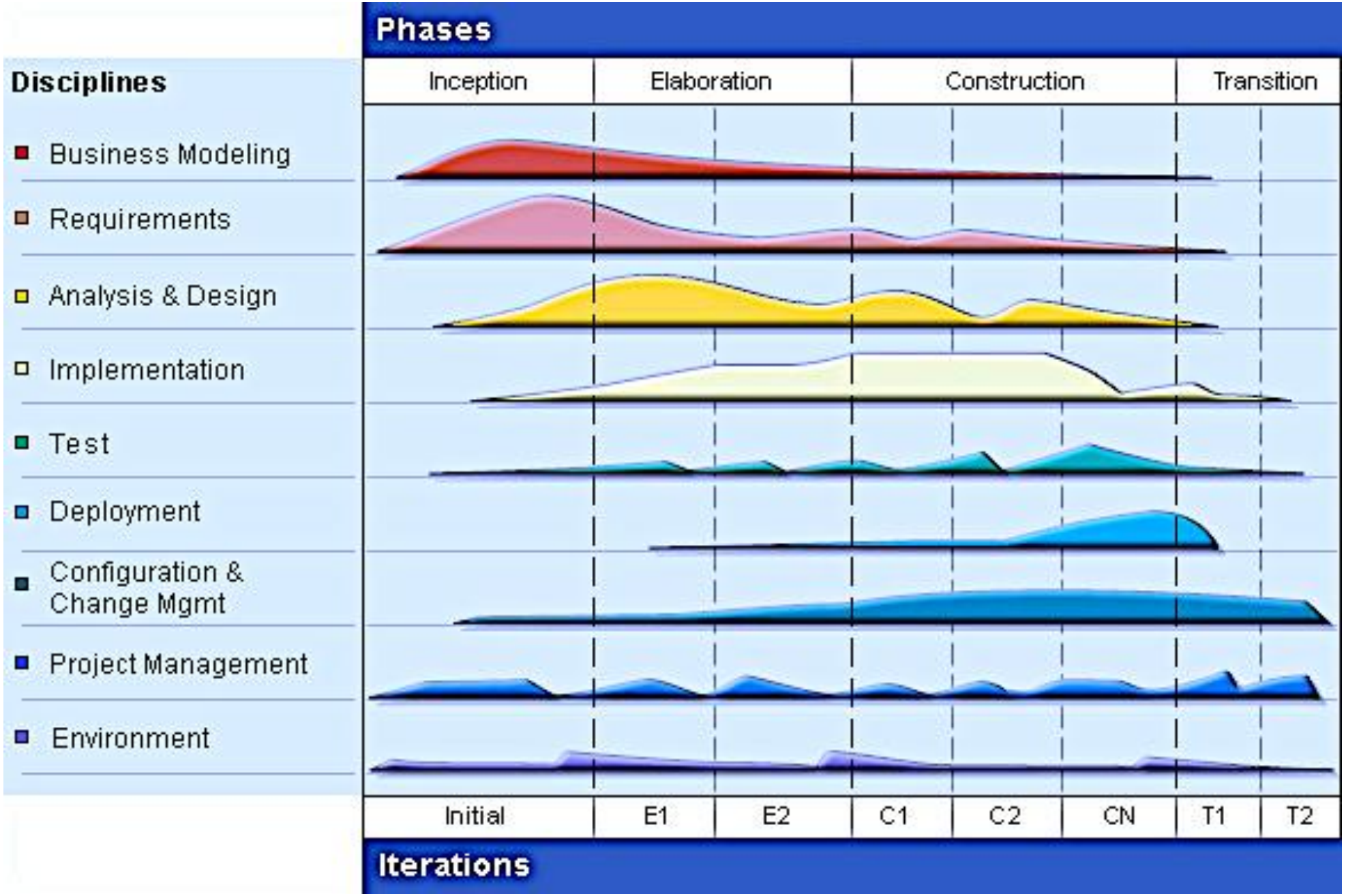
Rational Unified Process:

RUP Workflows:

- **Business Modeling:** It is about understanding how a business works by making simple diagrams that show:
  - **Who** is involved (like customers and employees).
  - **What** they do (like buying products or processing orders).
  - **How** these actions connect to each other.
- **Requirements:** Identify who will use the system (actors) and create use cases to outline what the system needs to do.
- **Analysis and Design:** Create a design model that shows how the system will be built. This includes different models like:
  - **Architectural models:** Overall structure of the system.
  - **Component models:** Breakdown of the system into parts.
  - **Object models:** Representation of data and its relationships.
  - **Sequence models:** How different parts of the system interact over time.
- **Implementation:** Build the system by developing the components and organizing them into sub-systems. Using automatic code generation tools can help speed up this process.
- **Testing:** Test the system as it is being developed, making sure to check for bugs. Once all parts are built, complete system testing is done to ensure everything works together.
- **Deployment:** Create the final version of the software, distribute it to users, and install it in their work environments.
- **Configuration and Change Management:** Manage any changes that need to be made to the system after it is deployed.
- **Project Management:** Oversee the entire development process to keep everything on track and within budget.
- **Environment:** Ensure the right software tools are available for the development team to work efficiently.

Rational Unified Process:

RUP Structure



Rational Unified Process:

Six Fundamental Practices of RUP:

1. **Develop Software Iteratively:** Build the software in small parts, focusing on the most important features first, based on what the customer needs.
2. **Manage Requirements:** Write down what the customer wants clearly and keep track of any changes. Check how these changes might affect the system before agreeing to them.
3. **Use Component-Based Architectures:** Organize the software into separate pieces (components) to make it easier to manage and develop.
4. **Visually Model Software:** Use simple diagrams (UML models) to show different aspects of the software, both how it looks (static) and how it behaves (dynamic).
5. **Verify Software Quality:** Make sure the software meets quality standards set by the organization.
6. **Control Changes to Software:** Keep track of any changes to the software using a system that manages these changes and procedures to ensure everything stays organized.

When RUP is not Suitable?

The RUP is not a suitable process for all types of development, e.g., embedded software development.

- Embedded systems often have strict hardware limitations and real-time requirements, which may not fit well with RUP's flexible and iterative approach.

When RUP Suitable? → Suitable When Iterative Development, Incremental Development and Reuse is Possible.

Advantages:

- Regular Feedback from Stakeholder
- Issues discovered early in project
- Deliver exactly what the customer want
- Use case Driven
- RUP reuses the components, and hence total time duration is less.

Dis-Advantages:

- Process may be too complex to implement
- Need Expert to fully adapt this process
- It is no longer widely-used and has been largely superseded by Agile and Scrum which are much more flexible and adaptive and easier to use

Software Process Improvement:

- Software Process Improvement is a way for software companies to make their development process better. The goal is to:
  - Increase the quality of their software,
  - Lower costs, and
  - Speed up development.

To do this, companies:

- **Study their current processes** to see how they work, and
- **Make changes** to switch development process, improve quality, reduce costs, and make development faster.

Software Process Improvement Cycle:

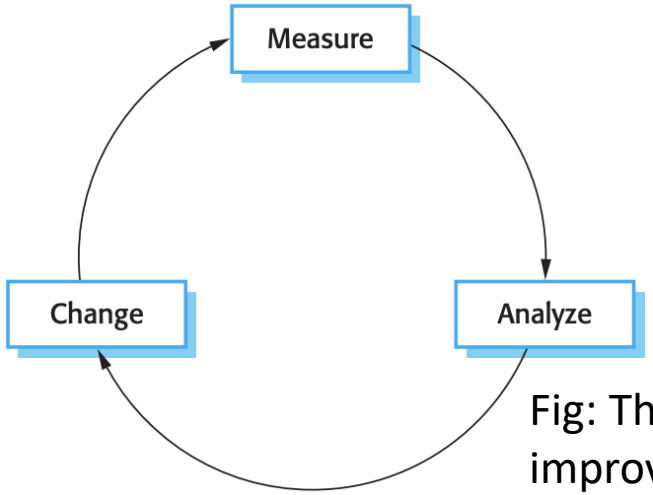


Fig: The process improvement cycle

The Software Process Improvement Cycle consists of three main steps:

- **Measure the Process:** Check how well the current software process is working to see if improvements are needed.
- **Analyze the Process:** Look for problems in the process to understand what isn't working well.
- **Make Changes:** Suggest and apply changes to fix the problems, then measure again to see if things are better.

Software Process Improvement Approaches:

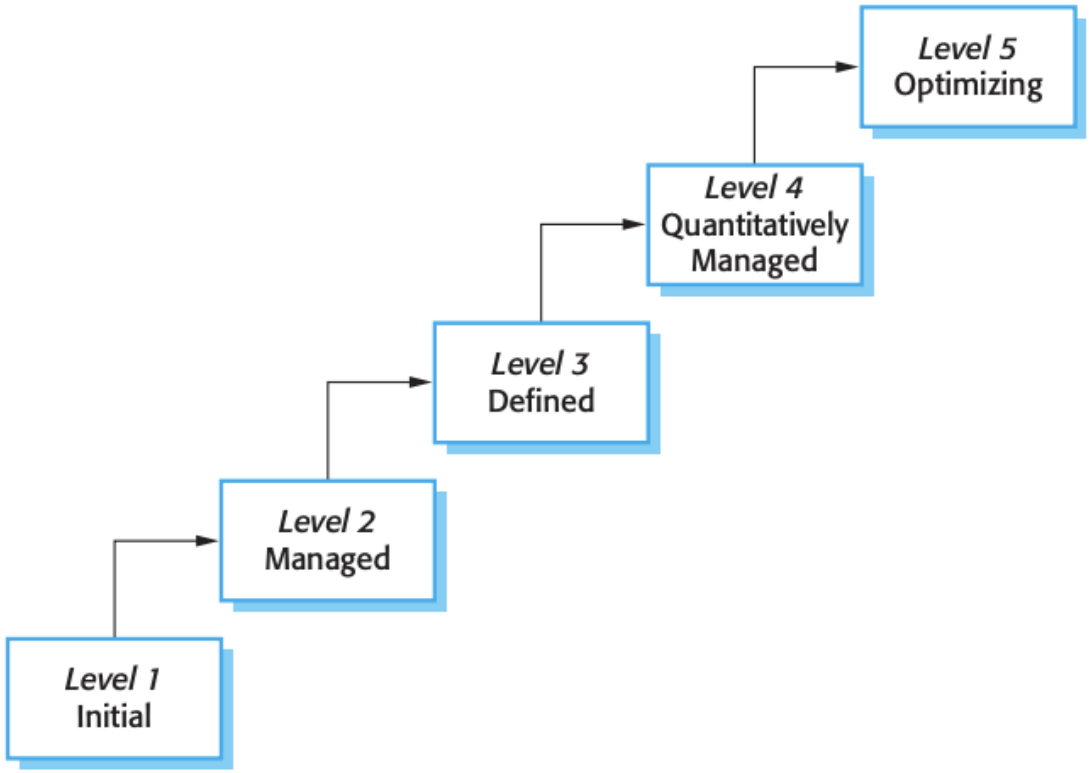
Two main types:

- **Process Maturity Approach:** This focuses on making project management and software development practices better. It measures how well an organization follows good technical and management practices in their software processes. Example: A company uses the Capability Maturity Model (CMM) to assess and improve its software development processes.
- **Agile Approach:** This emphasizes developing software in small, quick steps and reducing unnecessary steps in the process. Agile methods aim to deliver features rapidly and adapt quickly to changes in customer needs. Example: A development team uses Scrum, an Agile framework, to work on a new app.

Software Process Improvement:

The CMMI process improvement framework

- CMMI (Capability Maturity Model Integration) is a framework that helps organizations improve their s/w processes. It provides guidelines for developing and managing software and other projects more effectively.
- A CMMI assessment looks at how well an s/w development organization manages its processes and rates them on a six-point scale based on their maturity level. Here's a simple breakdown of the levels:



L1: Incomplete: Some goals for the process aren't met. There's no structure in place because the process isn't fully developed.

- Goals like → Meeting customer requirement, completing project on time, delivering product without bugs

- **Eg:** A startup begins developing a software app but skips defining clear requirements. As a result, they struggle to meet user needs, and the project lacks structure.

L2: Performed: The process meets its goals, and the team knows what work needs to be done.

- **Eg:** A small team develops a website and completes all tasks as planned. They meet the project goals, but they don't have a formal process for tracking progress.

L3: Managed: The process meets its goals, and there are clear policies and plans for how and when to use it. Resources are managed, and progress is monitored.

- **Eg:** A software company has a clear plan for its projects. They document tasks, set timelines, and assign resources. They regularly check progress to ensure the project stays on track.

Software Process Improvement:

The CMMI process improvement framework

L4: Defined: The organization standardizes its processes. Each project uses a tailored version of established processes, and data is collected to improve future processes.

- **Eg:** A large organization standardizes its software development process. Each project uses a common framework but can adjust it to fit specific needs. They also collect data on past projects to improve future ones.

L5: Quantitatively Managed: The organization uses data and statistics to monitor and control processes effectively.

- **Eg:** A company uses data to analyze how long tasks take and how many bugs are found in their software. They make decisions based on this data to improve efficiency and quality.

L6: Optimizing: The highest level where the organization continuously improves processes based on data and trends, adapting to meet changing business needs.

- **Eg:** A tech company continuously reviews their processes and uses data to make ongoing improvements. They adapt their software development methods based on trends in user feedback and changing market demands.