Software Engineering

Unit 2:
Software Development Process Model

Compiled by: Sagar Rana Magar

Course Outline

- Software Process
- Software Process Model
  - The Waterfall model
  - Evolutionary Development
  - Component Based Software Engineering (CBSE)
  - Process Iteration
    - Incremental Delivery
    - Spiral Development
  - Rapid Software Development
    - Agile Method
    - Extreme Programming
    - Rapid Application Development
    - Software Prototyping
  - Rational Unified Process (RUP)
  - Computer Aided Software Engineering (CASE)
    - Overview of CASE Approach
    - Classification of CASE tools

Software Process

- A software process is the set of activities and associated results that produce a software product.
- It is a coherent set of activities for specifying, designing, implementing & testing software system.
- There are four basic key process activities which are common to all software processes:
  1. **Software Specifications**
     - Detailed description of a software system to be developed with its requirements.
  2. **Software Development**
     - Designing, programming, documenting, testing, and bug fixing is done.
  3. **Software Validation**
     - Evaluation software product is done to ensure that the software meets the business requirements as well as the end users needs.
  4. **Software Evolution**
     - Timely updating it for various reasons.

Software Process Model

- A software process model is an abstract representation of a process.
- It presents a description of a process from some particular perspective.
- The model specifies the stage and order of a process.
- So, it is representation of the order of activities of the process and the sequence in which they are performed.
- The goal of a software process model is:
  - to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.
- Examples of process perspectives are:
  - **workflow- perspective**: sequence of activities
  - **Data-flow perspective**: information flow
  - **Role/action perspective**: who does what

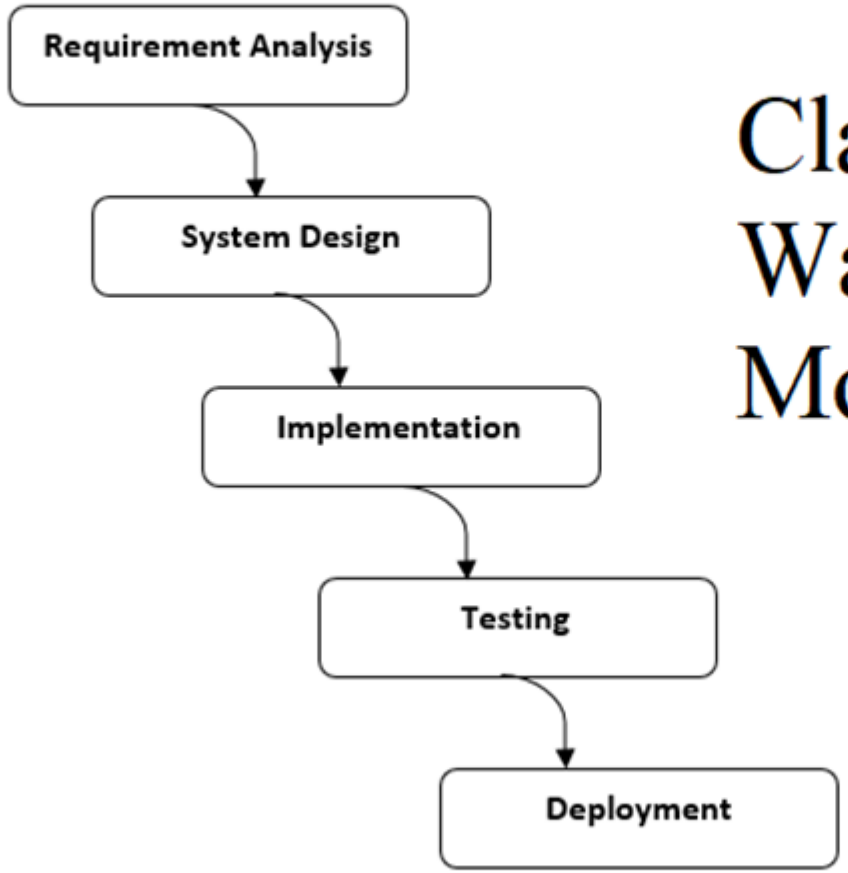
Software Process Model

- The most popular and important SDLC models are as follows:
  - Waterfall model
  - Incremental model
  - Iterative model
  - Prototype model
  - RAD model
  - Agile model
  - Spiral model
  - V model

Waterfall Model

- The waterfall model is a **sequential, plan driven-process**
  - where you must plan & schedule all your activities before starting the project.
- The waterfall model is a breakdown of project activities into linear sequential phases,
  - where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks.
- Each activity in the waterfall model is represented as a separate phase arranged in linear order.
- It is inflexible, it can't adapt to changes.
- There is no way to see or try the software until the last phase.

Waterfall Model



Classical Waterfall Model

Waterfall Model Application

- Some situations where the use of Waterfall model is most appropriate are:
  - Requirements are very well documented, clear and fixed.
  - Product definition is stable.
  - Technology is understood and is not dynamic.
  - There are no ambiguous requirements.
  - The project is short.

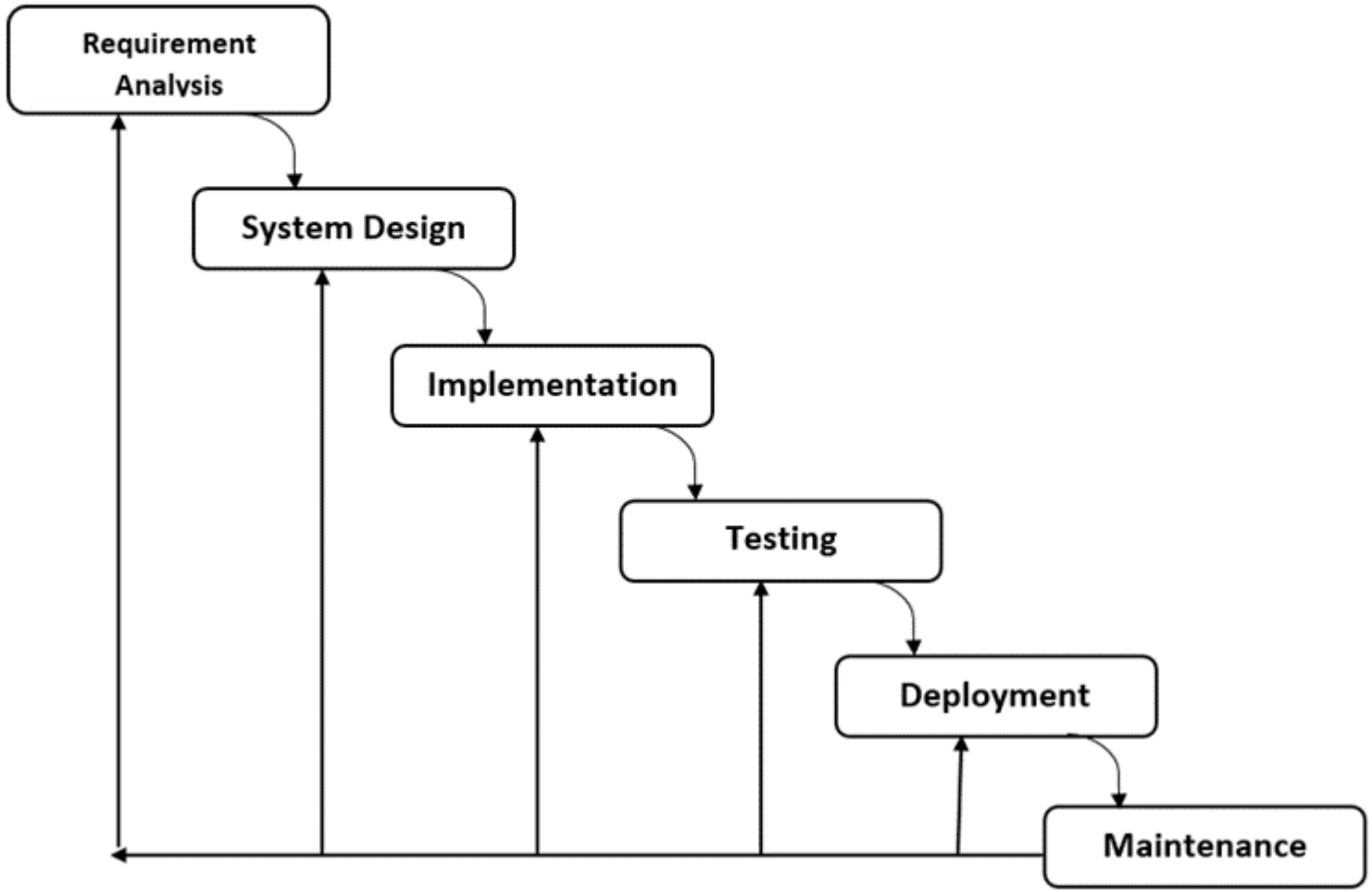
Advantages of Waterfall Model

- As each phase of development proceeds in strict order, it allows for departmentalization and control.
- Some other advantages of the Waterfall Model are as follows:
  - Simple and easy to understand and use
  - Easy to manage due to the rigidity of the model.
  - Each phase has specific deliverables and a review process.
  - Phases are processed and completed one at a time.
  - Works well for smaller projects where requirements are very well understood.
  - Clearly defined stages.
  - Easy to arrange tasks.

Drawbacks of Waterfall Model

- The main drawback of the waterfall model is
  - the difficulty of accommodating change after the process is underway.
- Therefore, this model is only appropriate when
  - the requirements are well-understood, and
  - changes will be fairly limited during the design process.
- In principle, a phase has to be completed before moving onto next phase.

Iterative Waterfall Model



Evolutionary Model

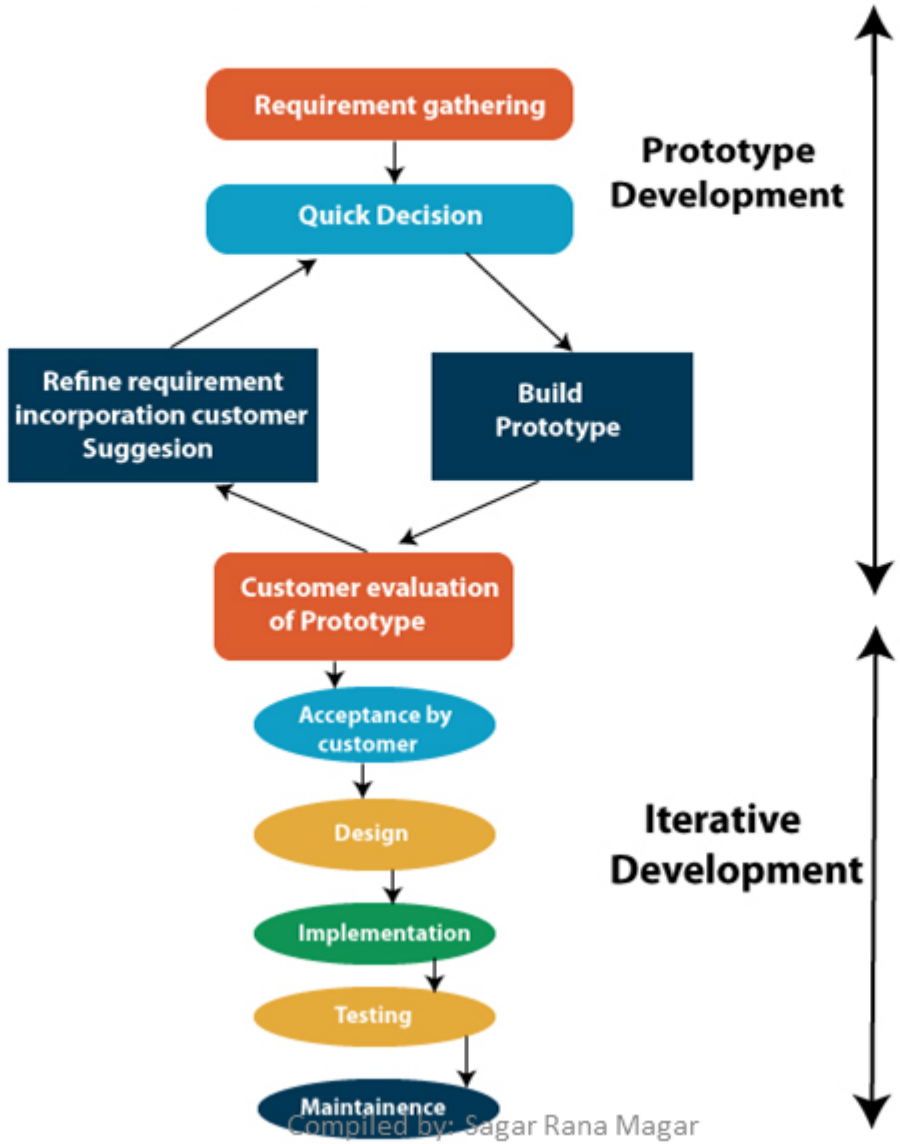
Prototype

- A prototype is an early sample, model or release of a product created to test a concept or process.
- Typically, a prototype is used to evaluate a new design to improve the accuracy of analysts and system users.
- The purpose of a prototype is to have a tangible model of the solutions to the problems already defined and discussed by the designers during the concept/idea stage.
- Prototypes allow designers to validate their concepts by putting an early version of the solution in front of real users and collecting feedback as quickly as possible.

Prototyping Model

- Prototyping is defined as the process of developing a working replication of a product or system, **prototype**, that has to be engineered.
- This model is used when the customers do not know the exact project requirements beforehand.
- In this model, a prototype of the end product is
  - first developed, tested and refined as per customer feedback repeatedly
  - till a final acceptable prototype is achieved which forms the basis for developing the final product.
- Prototyping is not a standalone, complete development methodology,
  - but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc).

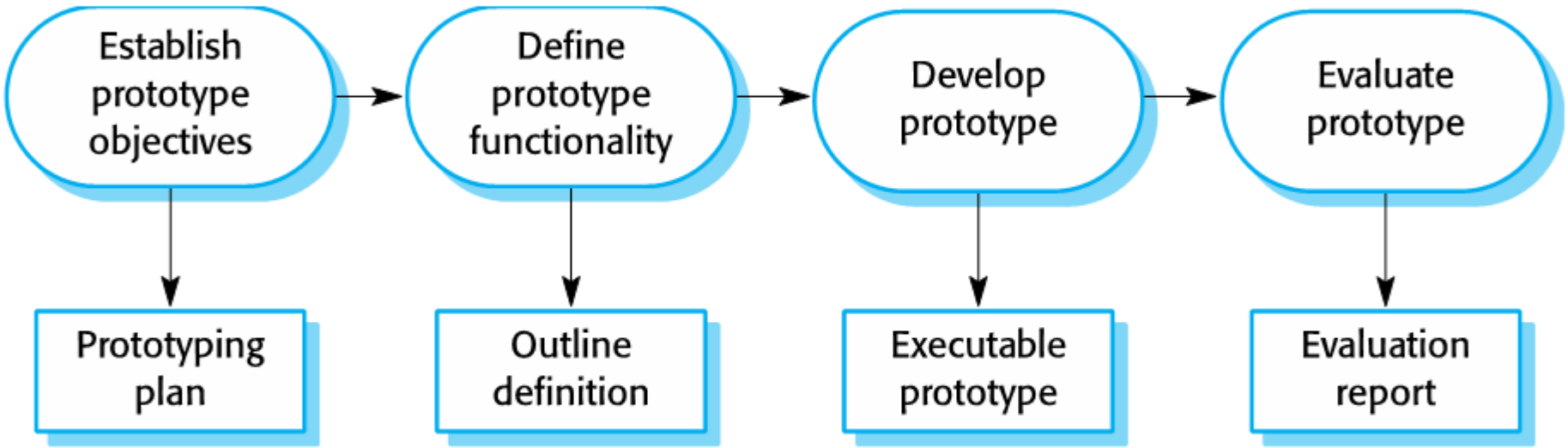
Prototyping Model



Benefits of Prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

Process of Prototype Development



Types of Prototyping

- Based on the usability and development process of the prototype that we have developed, this prototyping model can be categorized into :
  - Throw-away Prototyping
  - Evolutionary Prototyping
  - Incremental Prototyping
  - Extreme Prototyping

Throw-away Prototyping

- Throwaway prototyping is also called as **rapid** or **close ended prototyping**.
- It is created to see how the suggestions will visually look eventually.
- The customer's feedback changes the requirement, and the prototype is rebuilt until the conditions are met.
- Once the actual requirements are understood,
  - the prototype is discarded, and
  - the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping

- Evolutionary Prototyping is also called **breadboard prototyping**.
- It is based on building actual functional prototypes with minimal functionality in the beginning.
- The prototype developed forms the heart of the future prototypes on top of which the entire system is built.
- Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted.
- It helps you to save time as well as effort.

Incremental Prototyping

- Incremental prototyping refers to
  - building multiple functional prototypes of the various sub-systems, and
  - then integrating all the available prototypes to form a complete system.
- In Incremental Prototyping, the model developed is divided into smaller prototypes.
  - These prototypes are further developed individually.
- Features are added to each prototype until the final product is achieved.
- Later, various prototypes are eventually combined into a single product.

Extreme Prototyping

- Extreme prototyping method is mostly used for web development.
- It is consists of three sequential phases.
  - Basic prototype with all the existing page is present in the HTML format.
  - You can simulate data process using a prototype services layer.
  - The services are implemented and integrated into the final prototype.

Advantages of Prototyping

- Due to user involvement, the product risk to fail decreases.
- As there is a basic model, users will have a better understanding of the final system.
- Less time and cost as the errors are known earlier, and corrections are made.
- User feedback helps in developing a better solution.
- It gives scope for innovations and designs.

Disadvantages of Prototyping

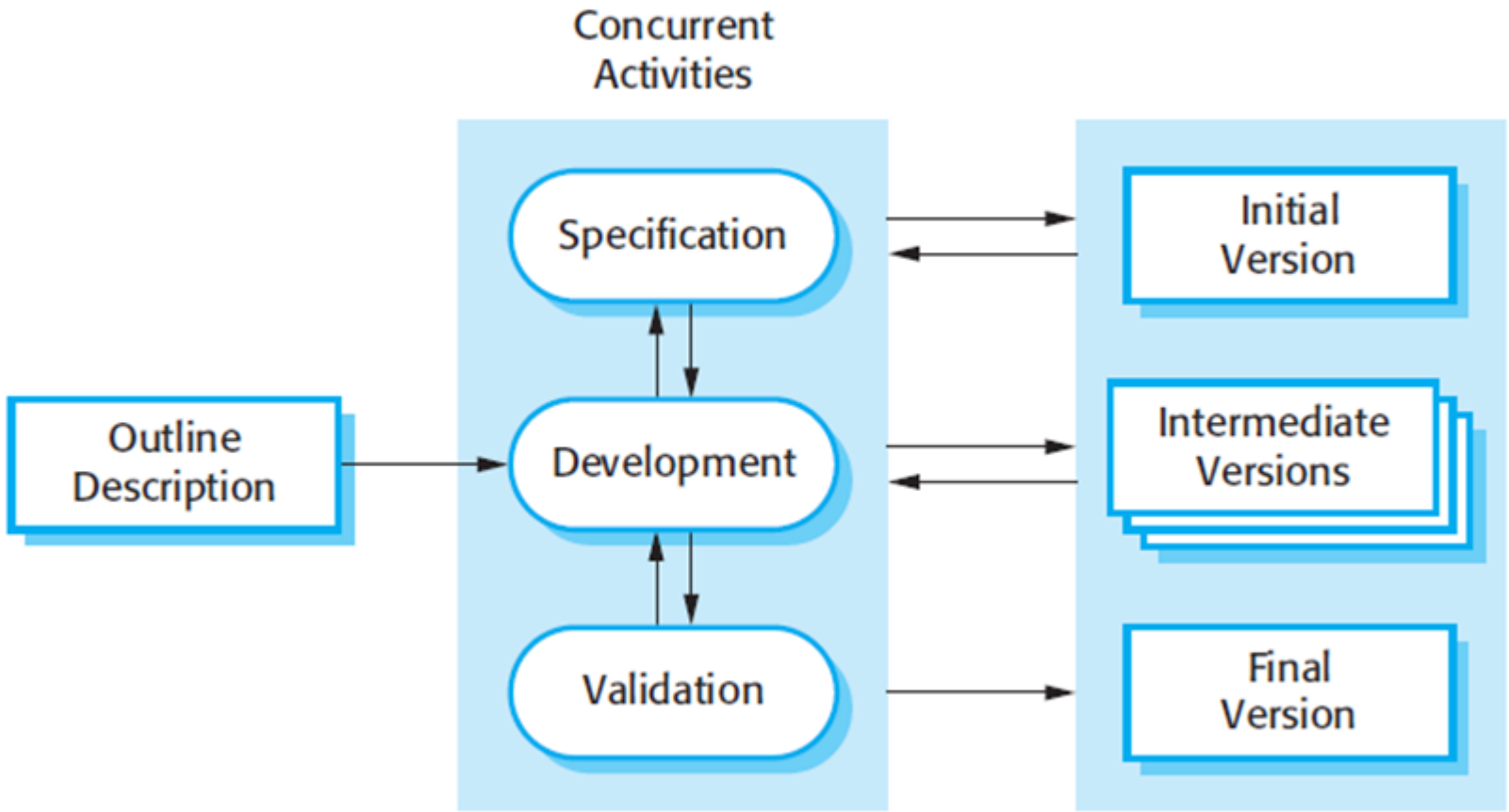
- There is insufficient requirement analysis as a complete prototype is in use.
- Confusion can be there between the prototype and the final product.
- This methodology may increase the system's complexity as the scope of the system expands beyond the actual plans.
- The time invested in a prototype can be wasted if it is not implemented efficiently.
- The improvement at every stage needs to be done quickly, or the prototype cannot be moved to the next step.

Process Iteration Model

Incremental Development

- Incremental development is based on the idea of
  - developing an initial implementation,
  - exposing this to user feedback, and
  - evolving it through several versions until an acceptable system has been developed.
- The activities of a process are not separated but interleaved with feedback involved across those activities.
- Each system increment reflects a piece of the functionality that is needed by the customer.
- Generally, the early increments of the system should include the most important or most urgently required functionality.
- The incremental model is great for projects that have loosely-coupled parts and projects with complete and clear requirements.

Incremental Development



Advantages of Incremental Development

1. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments.
   - These are part of the real system, so there is no relearning when the complete system is available.
2. Customers do not have to wait until the entire system is delivered before they can gain value from it.
   - The first increment satisfies their most critical requirements, so they can use the software immediately.
3. The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.
4. As the highest priority services are delivered first and later increments then integrated, the most important system services receive the most testing.
   - This means that customers are less likely to encounter software failures in the most important parts of the system.

Problems with Incremental Development

1. Iterative delivery is problematic when the new system is intended to replace an existing system.
    - Users need all of the functionality of the old system and are usually unwilling to experiment with an incomplete new system.
    - It is often impractical to use the old and the new systems alongside each other as they are likely to have different databases and user interfaces.
2. Most systems require a set of basic facilities that are used by different parts of the system.
    - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
3. The essence of iterative processes is that the specification is developed in conjunction with the software.
    - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.
    - In the incremental approach, there is no complete system specification until the final increment is specified.
    - This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.

Iterative Model

- Incremental process model is also know as **Successive version model**.
- In the Iterative model,
  - iterative process starts with a simple implementation of a small set of the software requirements, and
  - iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.
- An iterative life cycle model does not attempt to start with a full specification of requirements.
- Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements.
- This process is then repeated, producing a new version of the software at the end of each iteration of the model.

Iterative Model

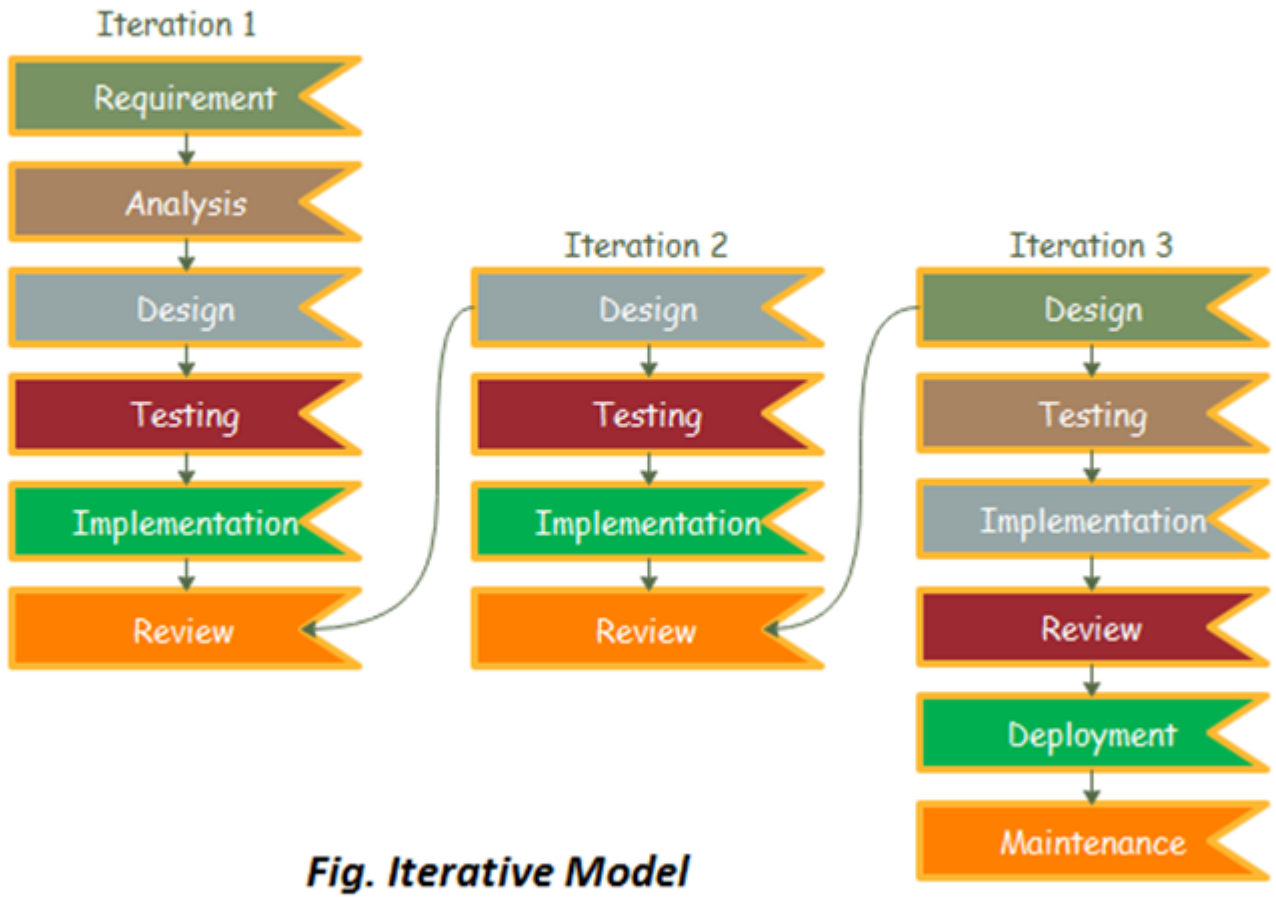
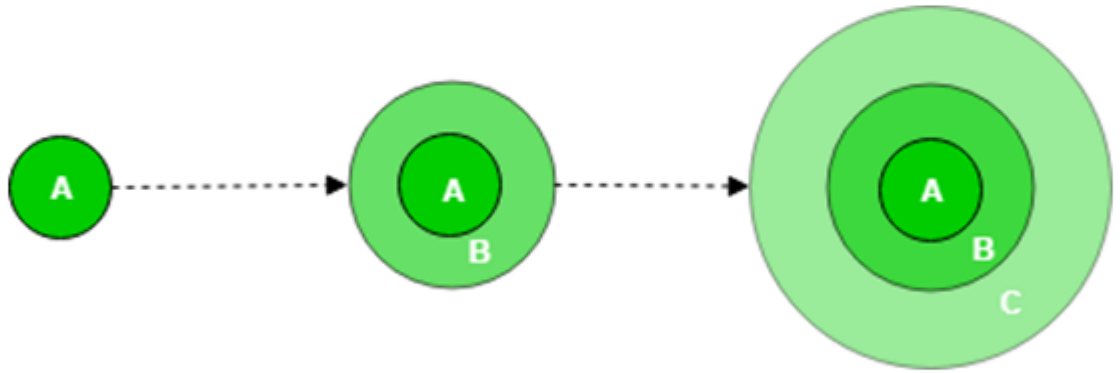


Fig. Iterative Model

Iterative Model

- First, a simple working system implementing only a few basic features is built and then that is delivered to the customer.
- Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.



- A, B, C are modules of Software Product that are incrementally developed and delivered.

When to use the Iterative Model?

- When requirements are defined clearly and easy to understand.
- When the software application is large.
- When there is a requirement of changes in future.
- A new technology is being used and is being learnt by the development team while working on the project.

Advantages of Iterative Model

- Testing and debugging during smaller iteration is easy.
- A Parallel development can plan.
- It is easily acceptable to ever-changing needs of the project.
- Risks are identified and resolved during iteration.
- Limited time spent on documentation and extra time on designing.

Disadvantages of Iterative Model

- It is not suitable for smaller projects.
- More Resources may be required.
- Design can be changed again and again because of imperfect requirements.
- Requirement changes can cause over budget.
- Project completion date not confirmed because of changing requirements.
- Management complexity is more.

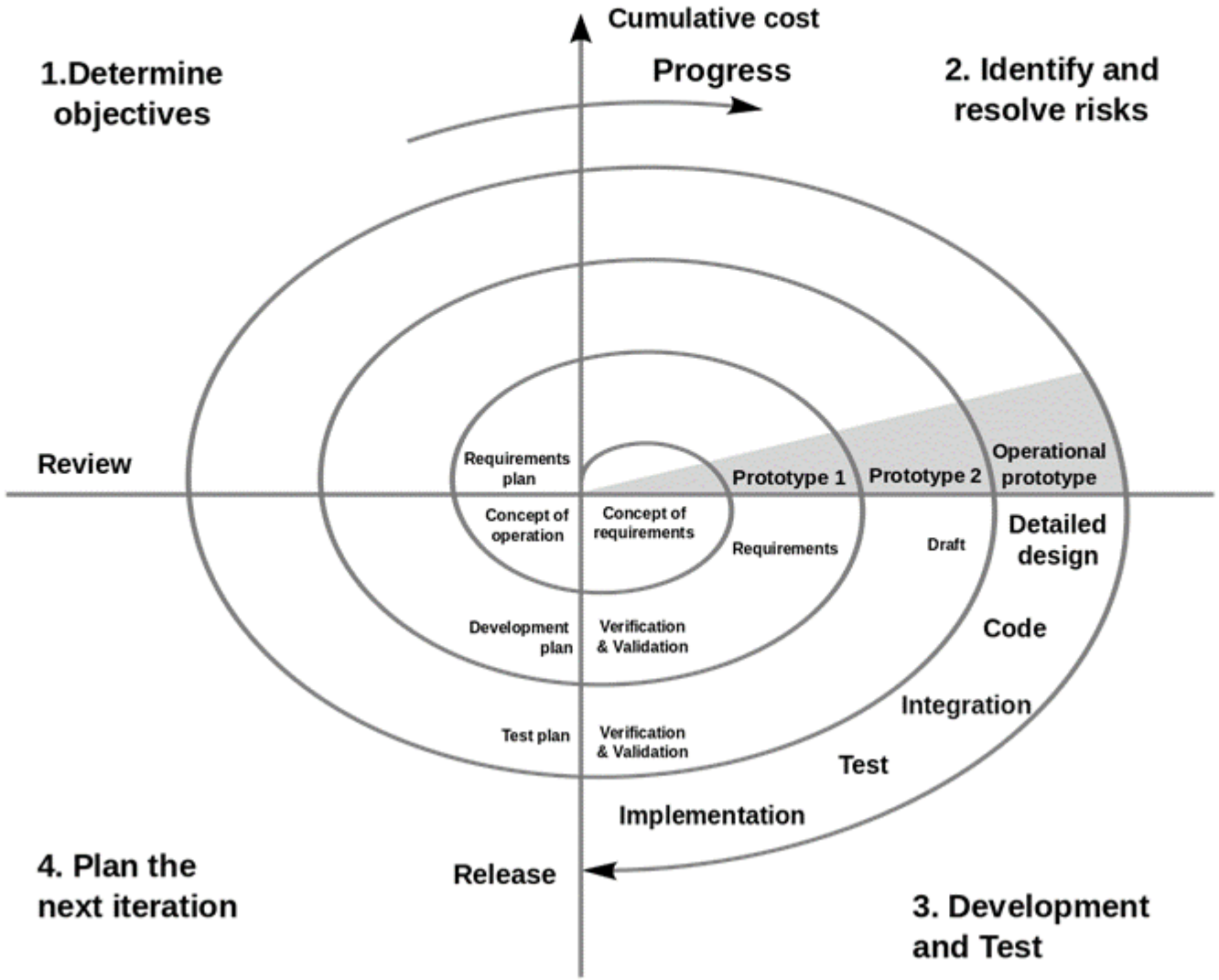
Spiral Development

- The spiral model is an evolutionary software process model that
  - couples the iterative development model with the controlled and systematic aspects of the linear sequential model.
- Spiral Model is a **risk-driven** software development process model.
- The spiral model is similar to the incremental development for a system, with more emphasis placed on risk analysis.
- The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.
- A software project repeatedly passes through these phases in iterations (called Spirals in this model).

Different Phases of Spiral Model

- The phase of the spiral model has four quadrants, and each of them represents some specific stage of software development.
- The four quadrants are listed below:
  - Planning objectives or identify alternative solutions
  - Risk Analysis
  - Development
  - Evaluation and plan for next Phase

Spiral Model



Risk Handling in Spiral Model

- Risk is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.
- The most important feature of the spiral model is handling these unknown risks.
  - Such risk resolutions are easier done by developing a prototype.
- The spiral model supports coping up with risks by
  - providing the scope to build a prototype at every phase of the software development.
- In Prototyping Model, risks must be identified completely before development of system
  - But in real life project risk may occur after the development work starts, so PM is not better.
- But, in each phase of the Spiral Model, the features of the product is analyzed,
  - and the risks at that point in time are identified and are resolved through prototyping.

When to use Spiral Model?

- When project is large
- When releases are required to be frequent
- When risk and costs evaluation is important
- When requirements are unclear and complex
- Significant changes are expected (research and exploration)
- When changes may require at any time
- When long term project commitment is not feasible due to changes in economic priorities

Advantages of Spiral Model?

1. **Risk Handling:**
   - it can analyze risk as well as handling risks at each phase of development
2. **Good for large projects:**
   - It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:**
   - Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:**
   - Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

Disadvantages of Spiral Model?

1. **Complex:**
   - The Spiral Model is much more complex than other SDLC models.
2. **Expensive:**
   - Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:**
   - The successful completion of the project is very much dependent on Risk Analysis.
   - Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:**
   - As the number of phases is unknown at the start of the project, so time estimation is very difficult.

Re-Use Based Software Model

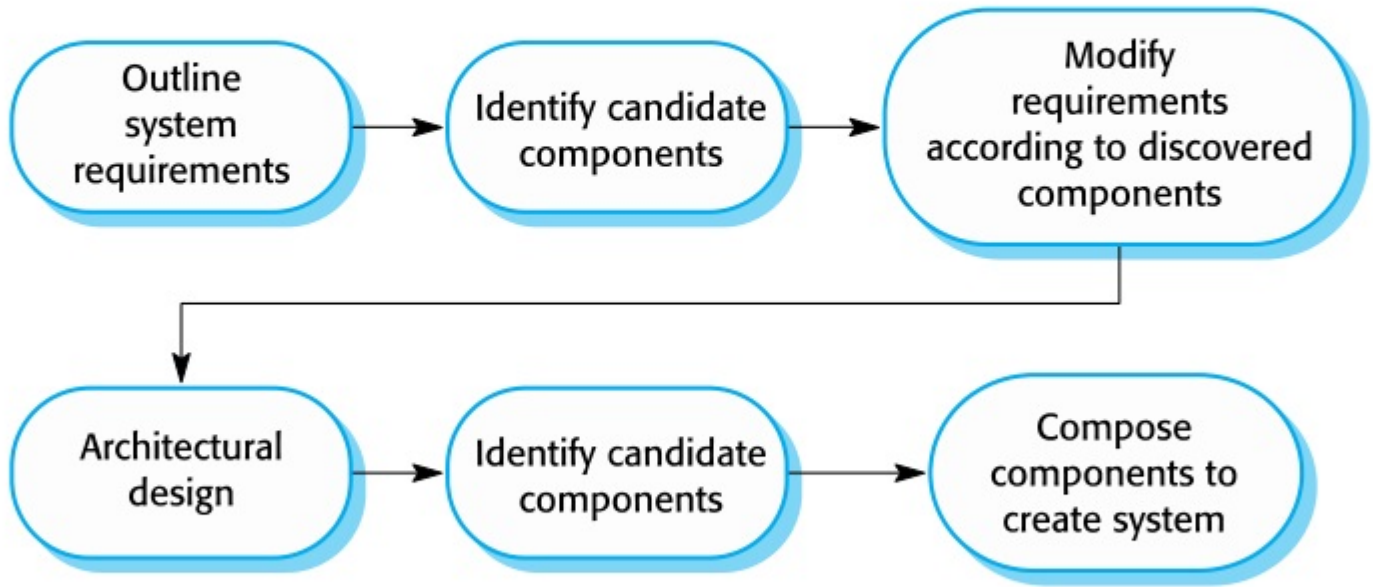
Component Based Software Engineering

- Component Based Software Engineering (CBSE) is also called **Component-based development**.
- It is a process model that focuses on the design and development of computer-based systems with the use of reusable software components.
- It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems.
- It emerges from the failure of the Object Oriented Designing to support effective reuse.

What is Component?

- A nontrivial, nearly independent, and replicable part of a system that fulfils a clear function in the context of a well-defined architecture.
- The component is an independent, executable entry. It can be made up of one or more executable objects.
- It does not have to be compiled before it is used with other components.
- Characteristics of component:
  - Standardized, Independent, Composable, Deployable, Documented

Component Based Software Engineering



Component Based S/W Engineering Process

- The process of computer-based software engineering is as follows:
  - First, it outlines all the system requirements.
  - Identify the components with component searching, Component selection, and component validation.
  - Modifying the requirements according to available functionality in the components.
  - Once again, search the components to find if there are better components that may meet the revised requirements.
  - Compose components to create the system.

Why Computer Based S/W Engineering?

- Save time and money
- Enhance the software quality
- Detect defects within the systems
- Increases quality, especially evolvalibilty and maintainability.

Advantages of CBSE

- Reusability
- Reduces development time and cost
- Increases Productivity
- Customization and Flexibility
- Reliability increases
- Maintainability

Rapid Software Development Model

Rapid Software Development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

Agile Method

- The meaning of Agile is Versatile or "able to move quickly or easily".
- Agile is the ability to create and respond to change.
- Agile is based on the **adaptive software development methods.**
  - where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed.
- Agile methodologies are approaches to product development that are aligned with the values and principles described in the Agile Manifesto for software development.
- Agile methodologies aim to deliver the right product, with incremental and frequent delivery of small chunks of functionality,
  - through small cross-functional self-organizing teams, enabling
  - frequent customer feedback and course correction as needed.

Agile Method

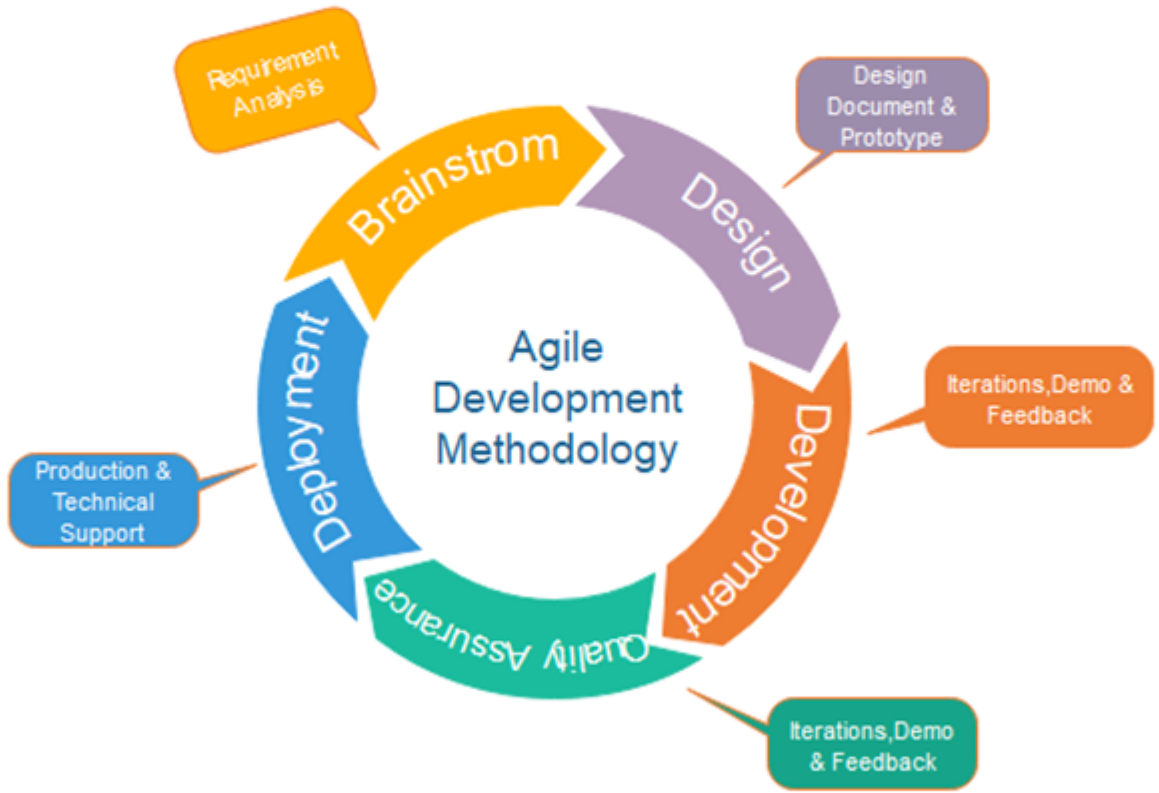


Fig. Agile Model

Rules of Agile Manifesto

- Individuals and interactions over processes and tools – The first value emphasizes teamwork and communication. We must understand that software development is a human activity and that the quality of interaction between people is vital. Tools are an important part of software development, but making great software depends much more on teamwork, regardless of the tools team may use.

- Working software over comprehensive documentation – Documentation has its place and can be a great resource or reference for users and coworkers alike. The main goal of software development, however, is to develop software that offers business benefits rather than extensive documentation.

- Customer collaboration over contract negotiation – Development teams must work closely and communicate with their customers frequently. By listening to and getting feedback, teams will understand what all stakeholders really want.

- Responding to change over following a plan – Changes are a reality in Software development, a reality that your Software process should reflect. A project plan must be flexible enough to change, as the situation demands.

12 Principle in Agile Manifesto in breif

1. Satisfying customers through early and continuous delivery of valuable work.
2. Breaking big work down into smaller tasks that can be completed quickly.
3. Recognizing that the best work emerges from self-organized teams.
4. Providing motivated individuals with the environment and support they need and trusting them to get the job done.
5. Creating processes that promote sustainable efforts.
6. Maintaining a constant pace for completed work.
7. Welcoming changing requirements, even late in a project.
8. Assembling the project team and business owners on a daily basis throughout the project.
9. Having the team reflect at regular intervals on how to become more effective, then tuning and adjusting behavior accordingly.
10. Measuring progress by the amount of completed work.
11. Continually seeking excellence.
12. Harnessing change for a competitive advantage.

Why Agile Method?

- Agile is about being responsive to the market and to the customer by
  - responding quickly to their needs and demands, and
  - being able to change direction as the situation demands.
- Agile methods attempt to
  - maximize the delivery of value to the customer, and
  - minimize the risk of building products that do not (or no longer) meet market or customer needs.

Pros of Agile Model

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Gives flexibility to developers.

Cons of Agile Method

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- Transfer of technology to new team members may be quite challenging due to minimum documentation.

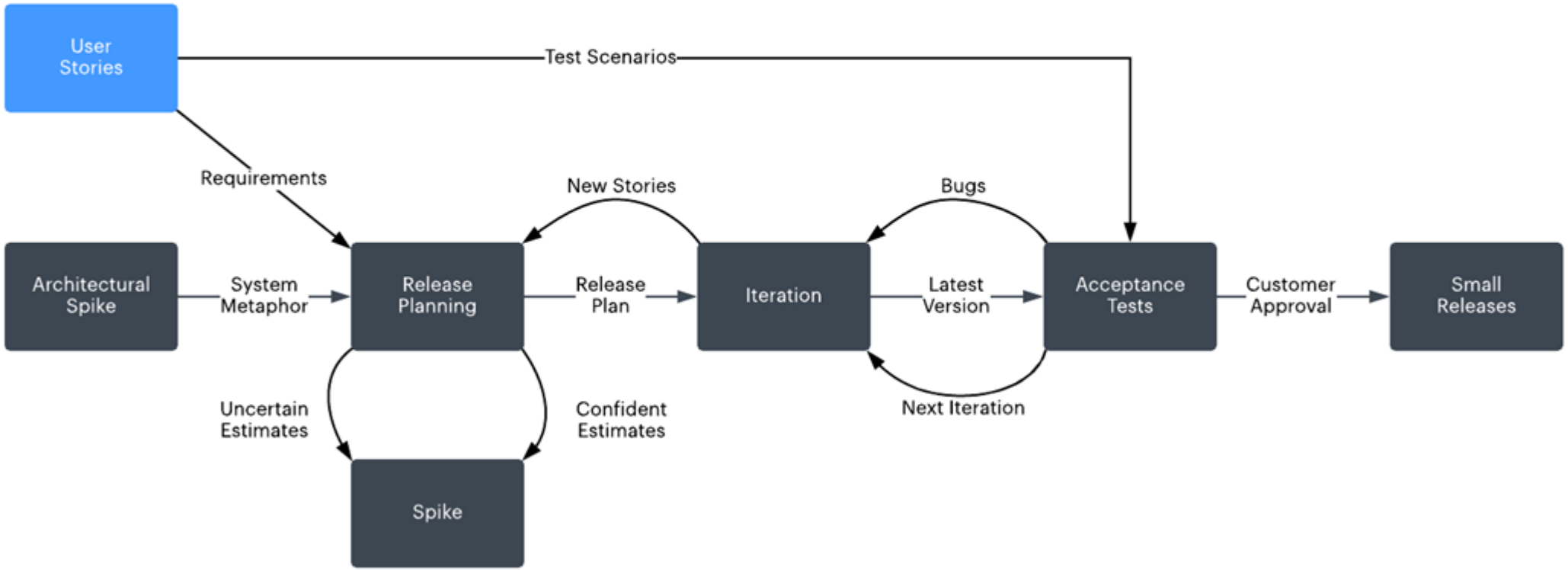
eXtreme Programming (XP)

- Extreme Programming (XP) is an agile software development framework
  - that aims to produce higher quality software, and higher quality of life for the development team.
- It is used to improve software quality and responsive to customer requirements.
- The extreme programming model recommends taking the best practices that have worked well in the past in program development projects to extreme levels.
- it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

eXtreme Programming (XP)

- XP begins with five values: Communication, feedback, Simplicity, Courage and Respect.
- High priority requirements are built first, no matter what is the complexity of that requirement.
- XP teams typically work in iterations which can be changed based on development requirements and current conditions.
- The main goal of every XP project is to provide a high-quality product to the customer in the shortest term.
- In XP projects, the team works on the software development along with the managers and the customers.
  - This measure is necessary to produce the final product quickly.

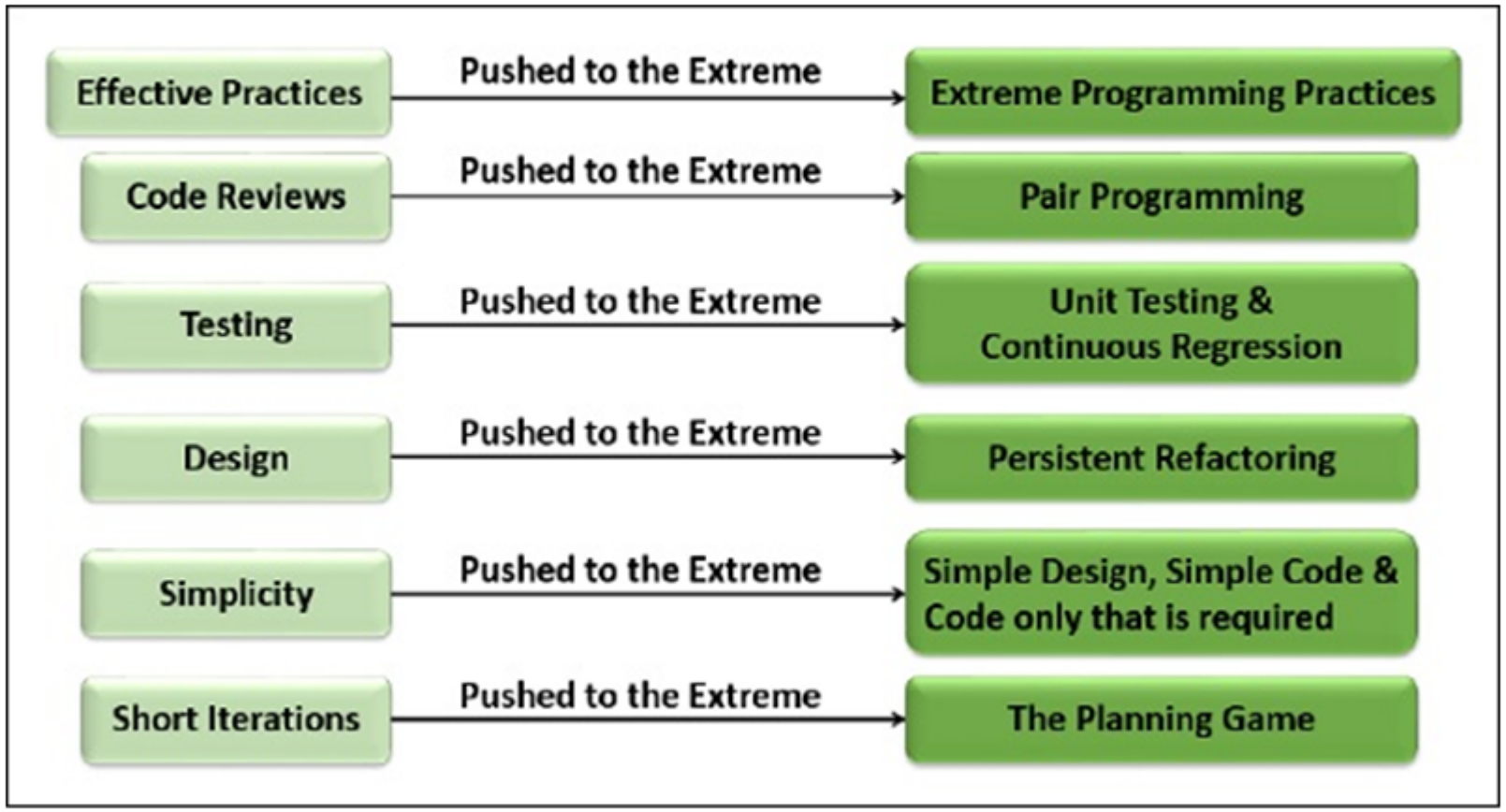
eXtreme Programming Methodology



Why is it called Extreme?

- Extreme Programming takes the effective principles and practices to extreme levels.
  - Code reviews are effective as the code is reviewed all the time.
  - Testing is effective as there is continuous regression and testing.
  - Design is effective as everybody needs to do refactoring daily.
  - Integration testing is important as integrate and test several times a day.
  - Short iterations are effective as the planning game for release planning and iteration planning.

Extreme Programming



XP Practices

1. The planning Game
2. Small Releases
3. Metaphor
4. Simple Design
5. Testing
6. Refactoring
7. Pair Programming
8. Collective Ownership
9. Continuous Integration
10. 40-Hour Work Week
11. On site Customer
12. Coding Standards

Where XP should be used?

- XP should be applied to systems that have no constant functionality features.
- Moreover, the XP method will succeed in situations where other approaches to software development will be useless.
- Extreme Programming is the best solution for the situation:
  - It is a good solution for risk issues.
  - Sometimes the risk of the project is great because the final product should be ready by a specific date.
  - It is even larger if you have no experience in such projects.
- This method of software development was designed especially for projects that should be performed immediately.
  - Speed is the largest advantage of the XP method.
- In addition, the high level of customer involvement into the project avoids the risk of low acceptance of the final product.

Rapid Application Development

- RAD is an **agile software development approach that focuses**
  - more on ongoing software projects and user feedback, and
  - less on following a strict plan.
- This methodology prioritizes rapid prototype releases and iterations.
- While regular plan-based methods require a rigid structure with specific requirements,
  - a RAD approach is based around flexibility and the ability to adapt alongside new knowledge.
- The key principle of the RAD process is
  - a reduction in planning to focus on a highly iterative design and construction process,
  - enabling teams to accomplish more in less time, without impacting client satisfaction.

Phases of RAD

- A rapid application development cycle consists of four steps:
  1. Define project requirements;
  2. Prototype;
  3. Rapid construction & feedback gathering; and
  4. Finalize product / implementation.

Define Requirements

- Rather than making you spend months developing specifications with users, RAD begins by **defining a loose set of requirements**.
- Loose because
  - permission to change requirements at any point in the cycle.
- Basically, developers gather the product's "gist."
- The client provides their vision for the product and comes to an agreement with developers on the requirements that satisfy that vision.

Prototype

- In this rapid application development phase, the developer's goal is to **build something that they can demonstrate to the client**.
  - This can be a prototype that satisfies all or only a portion of requirements (as in early stage prototyping).
- This prototype may cut corners to reach a working state, and that's acceptable.
- This is where the actual development takes place.
- Developers create prototypes with different features and functions as fast as they can.
- These prototypes are then shown to the clients who decides.
- This is normal, and the final product is only created during the finalization stage where the client and developer can both agree on the final product.

Receive Feedback

- In this stage, feedback on what's good, what's not, what works, and what doesn't is shared.
- Feedback isn't limited to just pure functionality, but also visuals and interfaces.
- With this feedback in mind, prototyping continues.
- These two steps are repeated until a final product can be realized that fits both the developers' and client's requirements.

Finalize Product

- Here, features, functions, and interface of the software are finalized with the client.
- Stability, usability, and maintainability are of paramount importance before delivering to the client.

Pros of RAD

- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features.

Rational Unified Process (RUP)

Rational Unified Process

- RUP is acronym for Rational Unified Process.
- It is an agile software development methodology.
- The fundamental purpose of the RUP is to provide a model for effectively implementing commercially proven approaches to development, for use throughout the entire software development life cycle.
- The Rational Unified Process (RUP) is iterative and agile.
  - Iterative because all of the process's core activities repeat throughout the project.
  - The process is agile because various components can be adjusted, and phases of the cycle can be repeated until the software meets requirements and objectives.

Phases of RUP

- The Rational Unified Process is not a concrete development model,
  - but rather is intended to be adaptive and tailored to the specific needs of your project, team, or organization.
- The Rational Unified Process is based on a few fundamental ideas,
  - such as the phases of development and the building blocks, which define who, what, when, and how development will take place.
- It divides the development process into four distinct phases
  - that each involve business modeling, analysis and design, implementation, testing, and deployment.
- The four phases are:
  - Inception, Elaboration, Construction and Transition

Phases of RUP

- Inception Phase
  - During the inception phase, the basic idea and structure of the project is determined.
  - The team will sit down and determine if the project is worth pursuing at all, based on
    - the proposed purpose of the project,
    - the estimated costs (monetary and time), and
    - what resources will be required to complete the project once the green light is given.

Phases of RUP

- Elaboration Phase
  - The purpose of the elaboration phase is to analyze the requirements and necessary architecture of the system.
  - The success of this phase is particularly critical,
    - as the final milestone of this phase signifies the transition of the project from low-risk to high-risk,
    - since the actual development and coding will take place in the following phase.

Phases of RUP

- Construction Phase
  - As the implementation of the software development life cycle,
    - the construction phase is when the coding and implementation of all application features will take place.
    - This period is also where integrations with other services or existing software should occur..

Phases of RUP

- Transition Phase
  - Easier thought of as deployment, the transition phase is when the finished product is finally released and delivered to customers.
  - However, the transition phase is more than just the process of deployment;
  - it must also handle all post-release support, bug fixes, patches, and so forth.

Pros of RUP

- Allows for the adaptive capability to deal with changing requirements throughout the development life cycle, whether they be from customers or from within the project itself.
- Emphasizes the need (and proper implementation of) accurate documentation.
- Diffuses potential integration headaches by forcing integration to occur throughout development, specifically within the construction phase where all other coding and development is taking place.

Cons of RUP

- Heavily relies on proficient and expert team members,
  - since assignment of activities to individual workers should produce tangible, pre-planned results in the form of artifacts.
- Arguably, RUP is a fairly complicated model.
  - Given the assortment of the components involved, including best practices, phases, building blocks, milestone criteria, iterations, and workflows, often proper implementation and use of the Rational Unified Process can be challenging for many organizations, particularly for smaller teams or projects.

Factors in choosing a Software Process Model

- Choosing the right software process model for your project is difficult.
- If you know your requirements well, it will be easier to select a model that best matches your needs.
- You need to keep the following factors in mind when selecting your software process model:
  - Project Requirements
  - Project Size
  - Project Complexity
  - Customer Involvement
  - Familiarity with Technology
  - Project Resources

CASE

- CASE is acronym for Computer Aided Software Engineering
- It is the implementation of computer facilitated tools and methods in software development to ensure a high-quality and defect-free software.
- CASE ensures a check-pointed and disciplined approach and helps designers, developers, testers, managers and others to see the project milestones during development.
- One of the best advantages of using CASE is the delivery of the final product, which is more likely to meet real-world requirements as it ensures that customers remain part of the process.

CASE Tools

- CASE tools are automated software packages that helps to automate activities in the SDLC.
- CASE tools aim to enforce an engineering-type approach to development of software systems.
- CASE tools range from simple diagramming tools to very sophisticated programs to document and automate most of the stages in the SDLC.

Types of CASE Tools

1. **Diagramming Tools:**
   - Help in diagrammatic and graphical representations of the data and system processes.
2. **Project Management Tools:**
   - Used in project planning, cost and effort estimation, project scheduling and resource planning.
3. **Quality Assurance tools:**
   - Focus on maintaining the quality of the software with constant monitoring, controlling and maintaining.
4. **Documentation Generators:**
   - Help in generating user and technical documentation as per standards.
5. **Code Generators:**
   - Aid in the auto generation of code, including definitions, with the help of the designs, documents and diagrams.
6. **Maintenance Tools:**
   - Tools required to maintain the software after deployment.

Advantages

- CASE tools improve quality and productivity of software.
- Produces system that more closely meet user needs and requirements.
- Produces system with excellent documentation.
- Tools are more effective for large scales system.
- Produces more flexible system.
- CASE tools reduce the time for the error correction and maintenance.

Disadvantages

- Very Complex.
- Not easily maintainable.
- Good quality CASE tools are expensive.
- May be difficult to use with the existing system.