Unit I

SDLC and Models

Software Development Life Cycle (SDLC)

- The software-development life-cycle is used to facilitate the development of a large software product in a systematic, well-defined, and cost-effective way.
- An information system goes through a series of phases from conception to implementation. This process is called the Software-Development Life-Cycle.
- The Software Development Lifecycle is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software which meets customer expectations. The software development should be complete in the pre-defined time frame and cost.
- SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC lifecycle has its own process and deliverables that feed into the next phase.

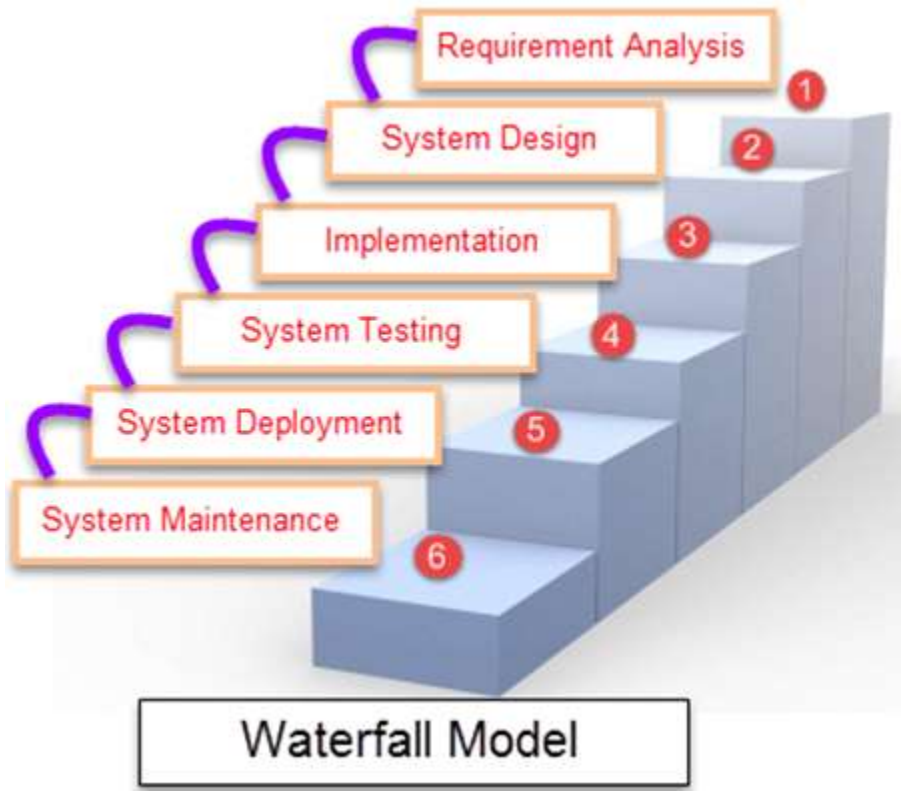
Phases of SDLC

These are:

- Project initiation and planning/Recognition of need/Preliminary investigation
- Project identification and selection/Feasibility study
- Project analysis
- System design
- Coding
- Testing
- Implementation
- Maintenance

Waterfall Model

- Waterfall Model is a sequential model that divides software development into different phases. Each phase is designed for performing specific activity during SDLC phase. It was introduced in 1970 by Winston Royce.



Different Phases of Waterfall Model in Software Engineering

Different phases	Activities performed in each stage
Requirement Gathering stage	•During this phase, detailed requirements of the software system to be developed are gathered from client
Design Stage	•Plan the programming language, for Example Java, PHP, .net •or database like Oracle, MySQL, etc. •Or other high-level technical details of the project
Built Stage	•After design stage, it is built stage, that is nothing but coding the software
Test Stage	•In this phase, you test the software to verify that it is built as per the specifications given by the client.
Deployment stage	•Deploy the application in the respective environment
Maintenance stage	•Once your system is ready to use, you may later require change the code as per customer request

Advantages of waterfall model:

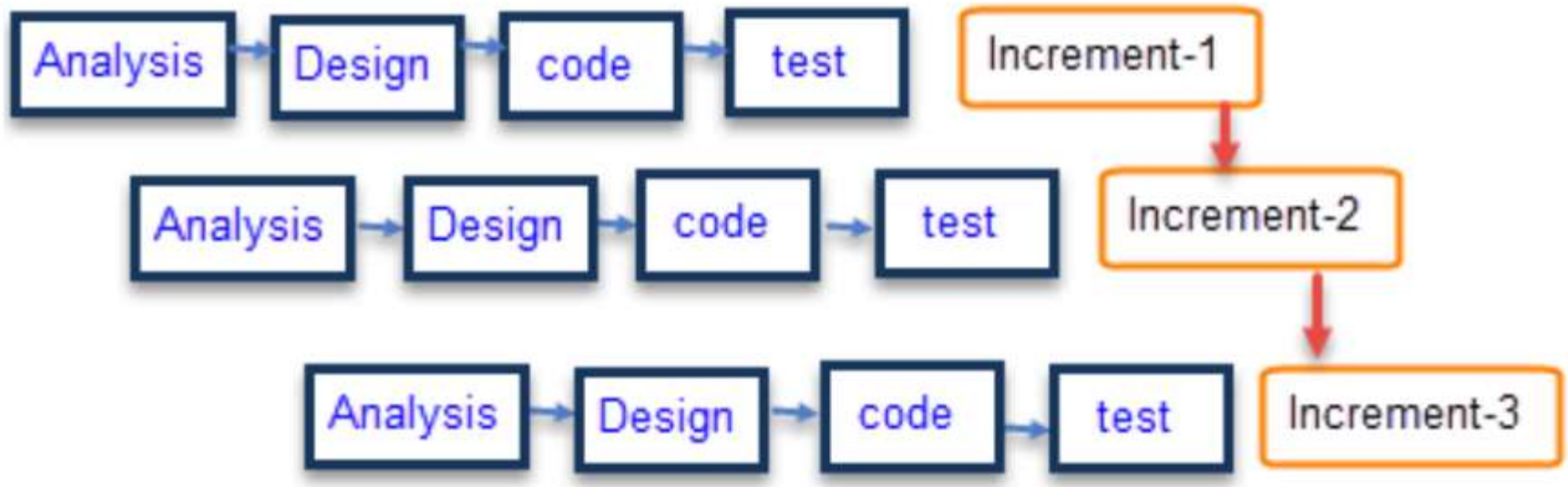
- The requirements are clearly and accurately stated, they remain unchanged throughout the entire project development;
- Detailed documentation of each development stage provides resistance to changes in human resources – a new developer can quickly get all the necessary information;
- Careful planning of the project development structure reduces the number of problematic issues;
- The start and end points for each phase are set, which makes it easy to measure progress;
- The tasks remain as stable as possible throughout the development process;
- It provides easy control and transparency for the customer due to a strict reporting system;
- Release date for the finished product, as well as its final cost can be calculated prior to development.

Disadvantages of the Waterfall Model:

- All requirements must be known prior to development, which greatly delays the project kickoff;
- Low flexibility level makes it difficult to make changes while developing, or even makes it completely impossible;
- There is a need for strict management and regular monitoring, so that the project will meet the deadline;
- The client does not have the opportunity to get acquainted with the system in advance, so he does not see the product until the moment of its completion;
- In case it becomes clear in the process of development that the product does not meet market requirements, there will be no room for changes.

Incremental Process Model

- Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.
- Each iteration passes through the **requirements, design, coding and testing phases**. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.



Incremental Model

Incremental Phases	Activities performed in incremental phases
Requirement Analysis	• Requirement and specification of the software are collected
Design	• Some high-end function are designed during this stage
Code	• Coding of software is done during this stage
Test	• Once the system is deployed, it goes through the testing phase

Advantages and Disadvantages of Incremental Model

Advantages	Disadvantages
•The software will be generated quickly during the software life cycle	•It requires a good planning designing
•It is flexible and less expensive to change requirements and scope	•Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle
•Throughout the development stages changes can be done	•Each iteration phase is rigid and does not overlap each other
•This model is less costly compared to others	•Rectifying a problem in one unit requires correction in all the units and consumes a lot of time
•A customer can respond to each building	
•Errors are easy to be identified	

When to use this :

- Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
- When Requirements are known up-front.
- When Projects having lengthy developments schedules.
- Projects with new Technology.

Evolutionary Process Models

- Evolutionary development is based on the idea of developing an initial implementation, exposing this to user comment and refining it through many versions until an adequate system has been developed (Figure 3.3). Specification, development and validation activities are interleaved with rapid feedback across activities.
- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

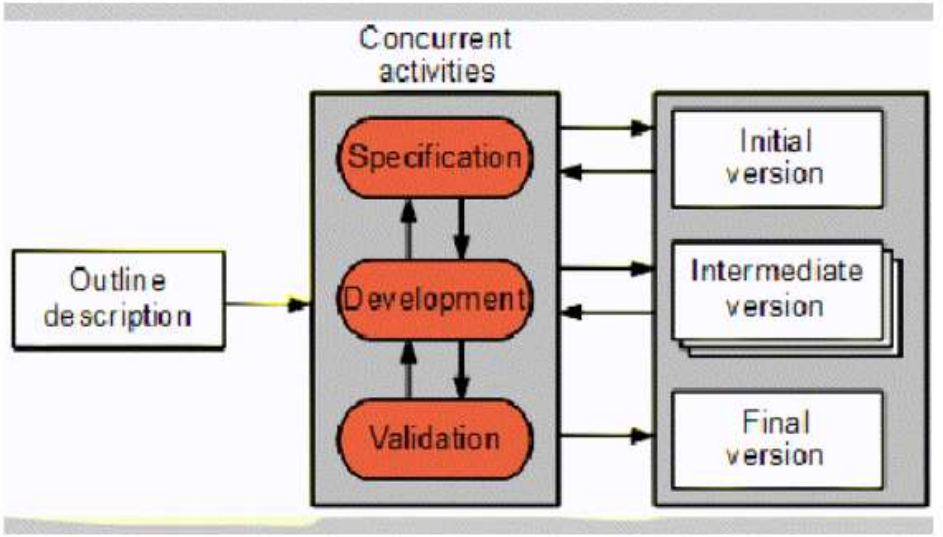


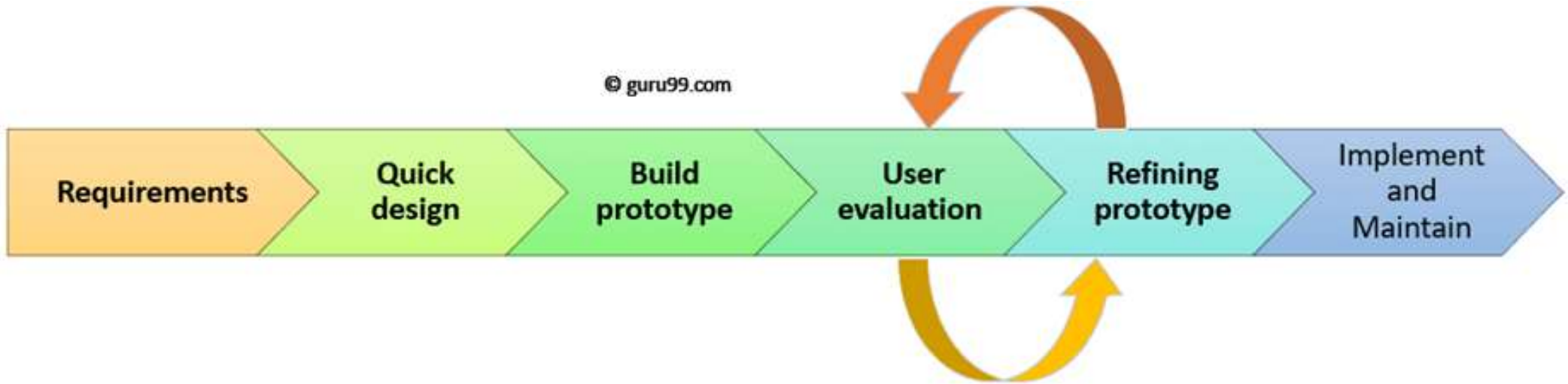
Figure 3.3. Evolutionary development.

Following are the evolutionary process models:

1. The prototyping model
2. The spiral model
3. Concurrent development model

The Prototyping model

- Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system.
- Software prototyping model works best in scenarios where the project's requirement are not known. It is an iterative, trial, and error method which take place between the developer and the client.


© guru99.com
Requirements
Quick design
Build prototype
User evaluation
Refining prototype
Implement and Maintain

Prototyping Model has following six SDLC phases as follow:

Step 1: Requirements gathering and analysis

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

Step 2: Quick design

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

Step 3: Build a Prototype

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

Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

Step 6: Implement Product and Maintain

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

Types of Prototyping Models

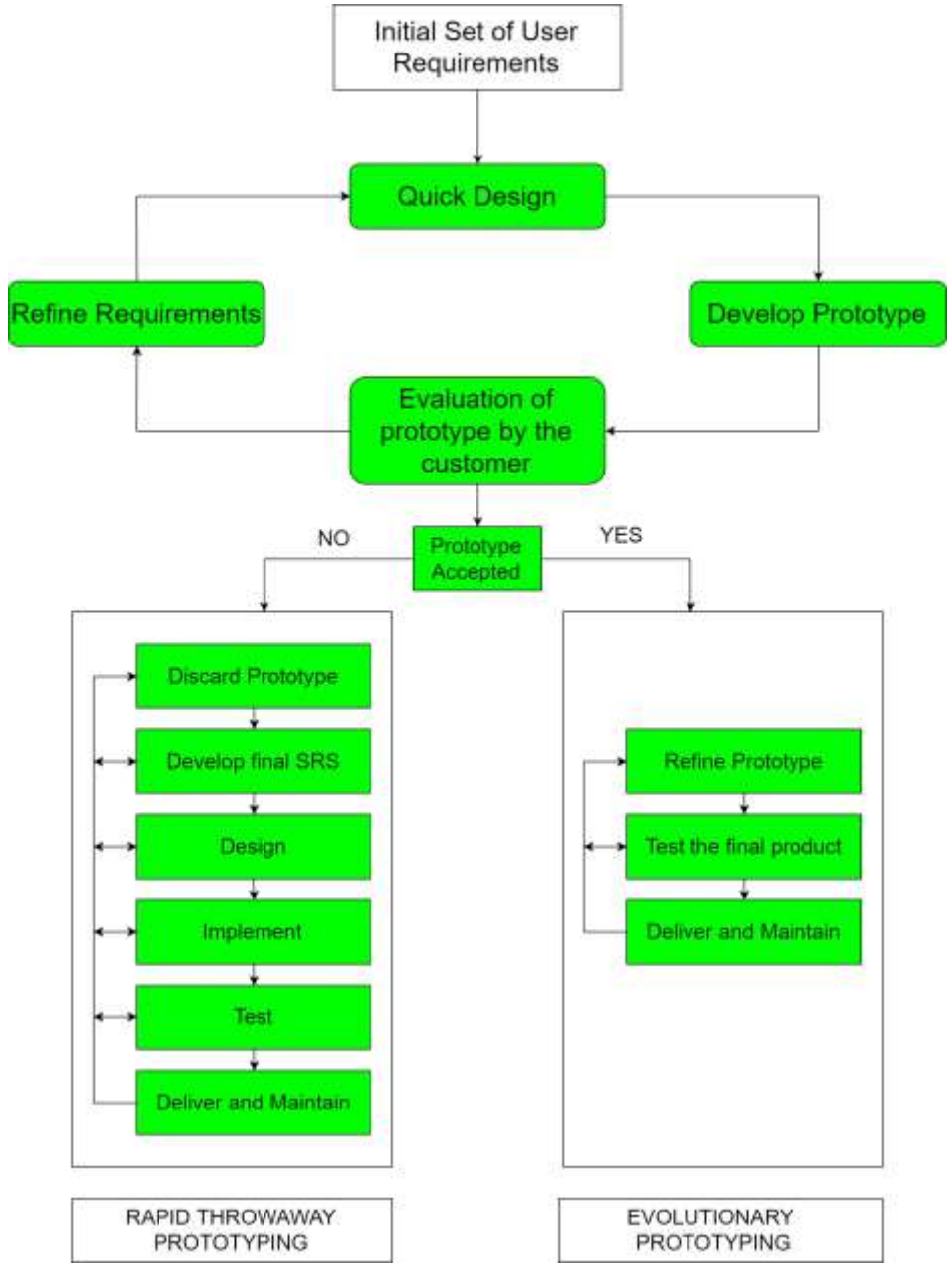
Four types of Prototyping models are:

- Rapid Throwaway prototypes
- Evolutionary prototype
- Incremental prototype
- Extreme prototype

- **Rapid Throwaway Prototype:**
- Rapid throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined.
- In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.
- **Evolutionary Prototyping:**
- 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. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating.
- This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

- **Incremental Prototyping**
- In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is helpful to reduce the feedback time between the user and the application development team.

- **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.


Initial Set of User Requirements
Quick Design
Refine Requirements
Develop Prototype
Evaluation of prototype by the customer
NO
Prototype Accepted
YES
Discard Prototype
Develop final SRS
Design
Implement
Test
Deliver and Maintain
Refine Prototype
Test the final product
Deliver and Maintain
RAPID THROWAWAY PROTOTYPING
EVOLUTIONARY PROTOTYPING

Advantages

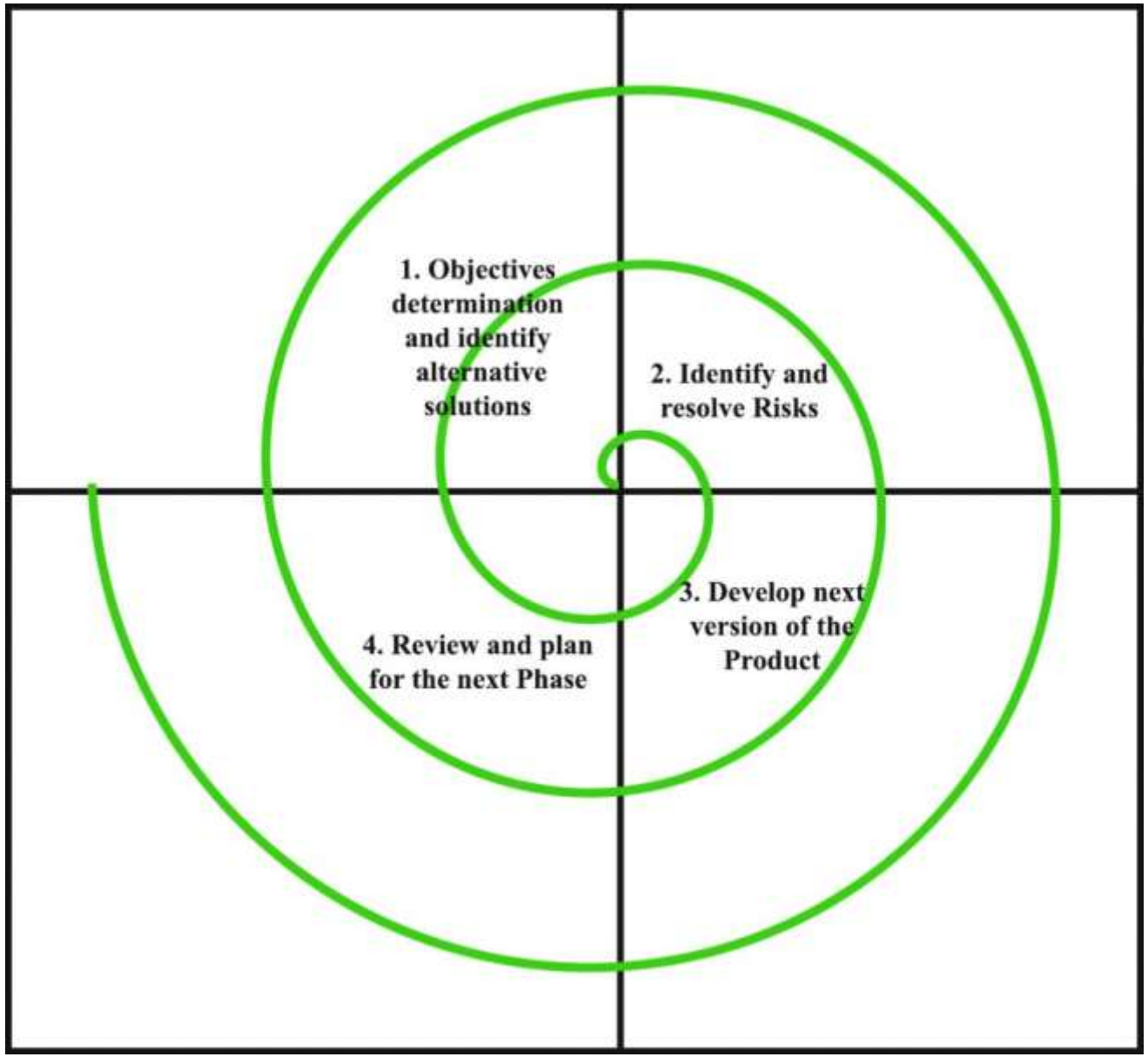
- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.

Disadvantages

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

Spiral Model

- **Spiral model** is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. **Each loop of the spiral is called a Phase of the software development process.** The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

The functions of these four quadrants are discussed below-

1.Objectives determination and identify alternative solutions: Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

2.Identify and resolve Risks: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.

3. Develop next version of the Product: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4. Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Risk Handling in Spiral Model

- A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. 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.
- **Prototyping Model** also support risk handling, but the risks must be identified completely before the start of the development work of the project. But in real life project risk may occur after the development work starts, in that case, we cannot use Prototyping Model. In each phase of the Spiral Model, the features of the product dated and analyzed and the risks at that point of time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

Advantages of Spiral Model: Below are some of the advantages of the Spiral Model.

- **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
- **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- **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:

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

Concurrent development model

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

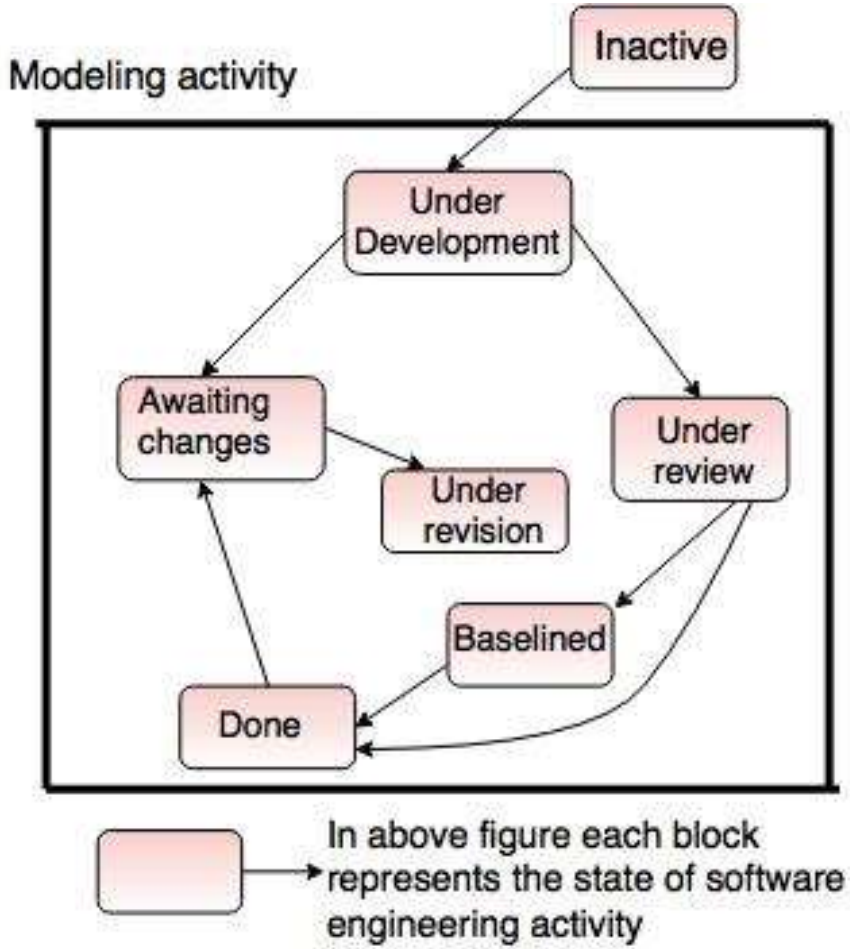


Fig. - One element of the concurrent process model

Advantages of the concurrent development model

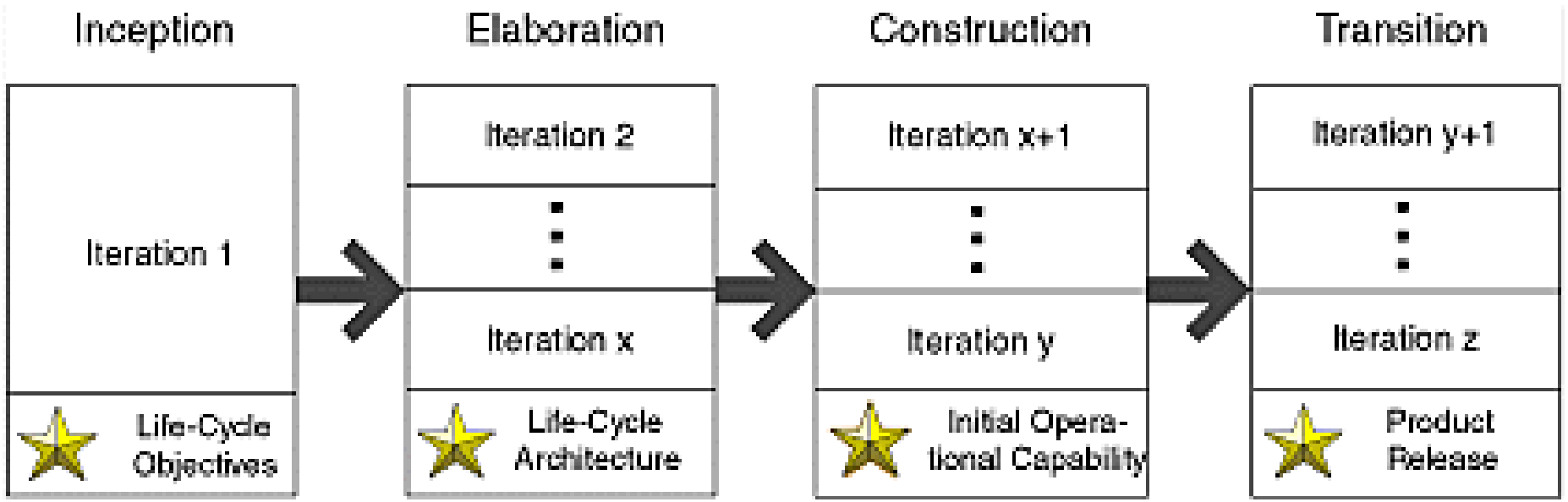
- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

Unified Process

- The **Unified Software Development Process** or **Unified Process** is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process(RUP). Other examples are OpenUP and Agile Unified Process.
- Unified process (UP) is an architecture-centric, use-case driven, iterative and incremental development process that leverages unified modeling language and is compliant with the system process engineering metamodel. Unified process can be applied to different software systems with different levels of technical and managerial complexity across various domains and organizational cultures.
- UP is also referred to as the unified software development process


Inception
Elaboration
Construction
Transition
Iteration 1
Iteration 2
Iteration x
Iteration x+1
Iteration y
Iteration y+1
Iteration z
Life-Cycle Objectives
Life-Cycle Architecture
Initial Operational Capability
Product Release

The UP defines the following four phases:

- **The Inception phase**, concluding with the Objective milestone, focuses on establishing the project's scope and vision; that is, establishing the business feasibility of the effort and stabilizing the objectives of the project.
- **The Elaboration phase**, concluding with the Architecture milestone, focuses on establishing the system's requirements and architecture; that is, establishing the technical feasibility of the effort and stabilizing the architecture of the system.
- **The Construction phase**, concluding with the Initial Operational Capability milestone, focuses on completing construction or building of the system.
- **The Transition phase**, concluding with the Product Release milestone, focuses on completing transitioning or deployment of the system to the user community.

Component-Based Development

- Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.
- The *component-based development model* incorporates many of the characteristics of the spiral model. It is evolutionary in nature [Nie92], demanding an iterative approach to the creation of software. However, the component based development model comprises applications from prepackaged software components.
- Modeling and construction activities begin with the identifi cation of candidate components. These components can be designed as either conventional software modules or object-oriented classes or packages 11 of classes.

- The component-based development model incorporates the following steps (implemented using an evolutionary approach):

1. Available component-based products are researched and evaluated for the application domain in question.

2. Component integration issues are considered.

3. A software architecture is designed to accommodate the components.

4. Components are integrated into the architecture.

5. Comprehensive testing is conducted to ensure proper functionality.

- The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits including a reduction in development cycle time and a reduction in project cost if component reuse becomes part of your organization’s culture

What is Agile Development?

- AGILE methodology is a practice that promotes **continuous iteration** of development and testing throughout the software development lifecycle of the project.
- The agile software development emphasizes on four core values.

✓Individual and team interactions over processes and tools

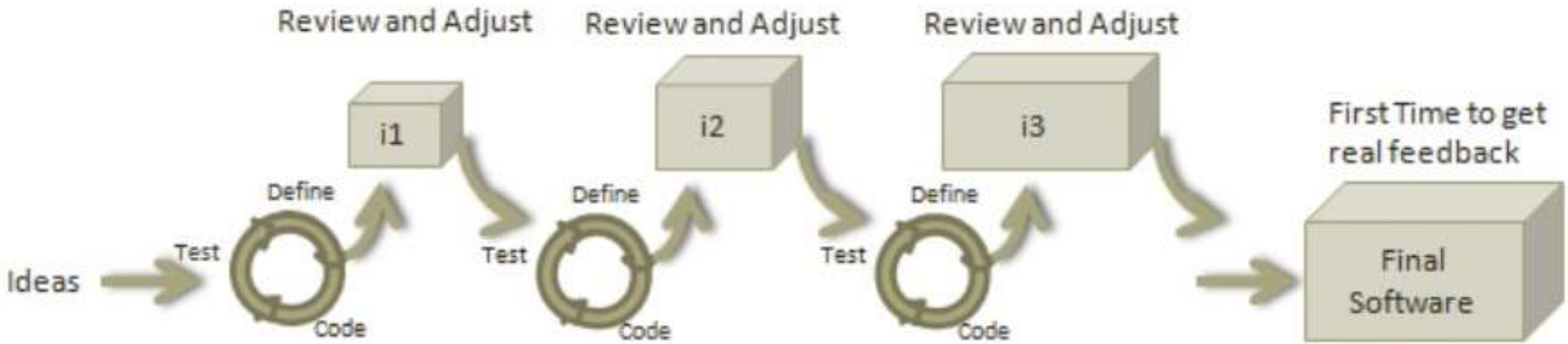
✓Working software over comprehensive documentation

✓Customer collaboration over contract negotiation

✓Responding to change over following a plan

- Agile is a software development methodology to build a software incrementally using short iterations of 1 to 4 weeks so that the development is aligned with the changing business needs.
- Agile is a software development methodology to build a software incrementally using short iterations of 1 to 4 weeks so that the development process is aligned with the changing business needs. Instead of a single-pass development of 6 to 18 months where all the requirements and risks are predicted upfront, Agile adopts a process of frequent feedback where a workable product is delivered after 1 to 4 week iteration.

Traditional Method



Agile Method

Principles of Agile Method

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

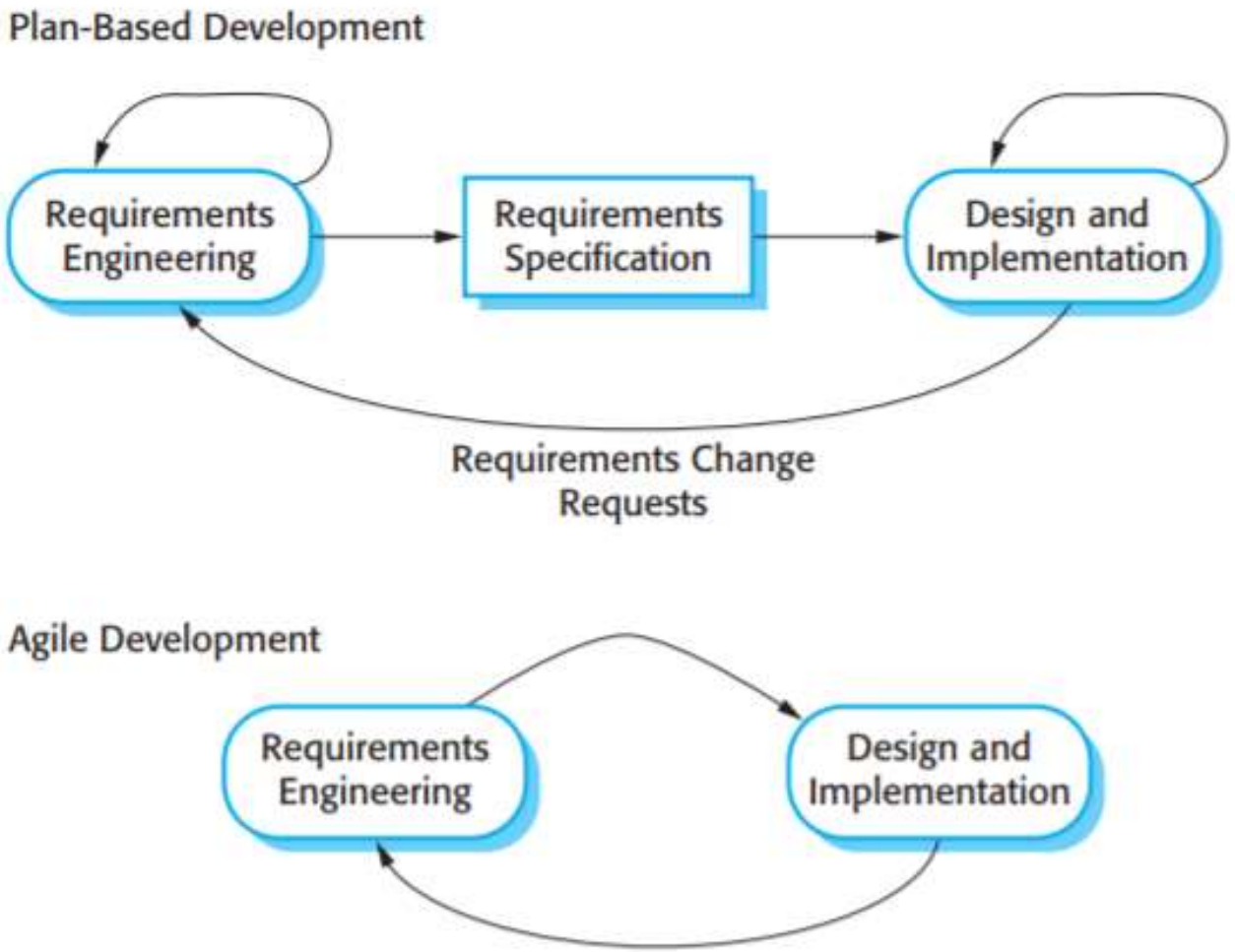
Plan-driven and agile development

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the plan and the software to reflect changing customer requirements.

Agile approaches to software development consider design and implementation to be the central activities in the software process. They incorporate other activities, such as requirements elicitation and testing, into design and implementation. In an agile approach, iteration occurs across activities. Therefore, the requirements and the design are developed together, rather than separately

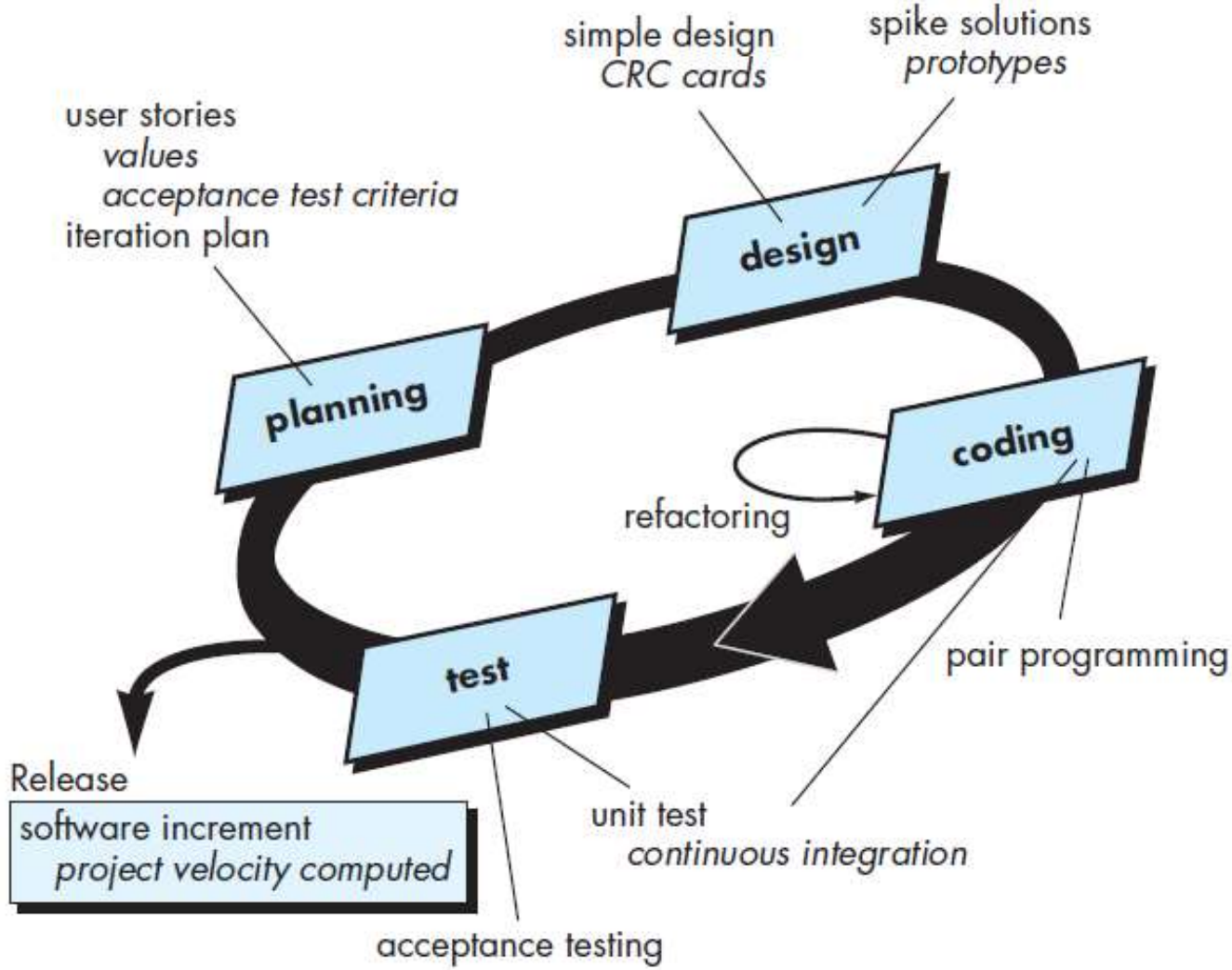
A plan-driven approach to software engineering identifies separate stages in the software process with outputs associated with each stage. The outputs from one stage are used as a basis for planning the following process activity. In a plan-driven approach, iteration occurs within activities with formal documents used to communicate between stages of the process. For example, the requirements will evolve and, ultimately, a requirements specification will be produced. This is then an input to the design and implementation process.

Plan-driven and agile specification



Extreme programming

- Extreme Programming technique is very helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system. It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented. The XP develops software keeping customer in the target.


simple design
CRC cards
spike solutions
prototypes
user stories
values
acceptance test criteria
iteration plan
design
planning
coding
refactoring
test
pair programming
Release
software increment
project velocity computed
unit test
continuous integration
acceptance testing

XP Core Values

The four core value of XP – communications, simplicity, feedback, and courage – drive its practices and project activities.

- **Communication:** One of the major causes of project failure has been a fact of open communication with the right players at the right time and at the right level. Effective communication involves not only documentation but also open verbal discussion. The practices and methods of XP are designed to ensure that open, frequent communication takes place.

- **Simplicity:** Developers have always advocated keeping solutions simple, they do not always follow their own advice. XP includes techniques to reinforce this principle and make it a standard way of developing systems.

- **Feedback:** Getting frequent and meaningful feedback is recognized as a best practice of software development. Feedback on functionality and requirements should come from the users, feedback on designs and code should come from other developers, and feedback on satisfying a business need should come from the client XP integrates feedback into every aspect of development.

- **Courage:** Developers always need courage to face the harsh choice of doing things right of throwing away bad code and starting over. But all too frequently they have not had the courage to stand up to a too right schedule, resulting in bad mistakes. XP practices are designed to make it easier to give developers the courage to “do it right.”

XP Practices

- **The Planning Process,** sometimes called the Planning Game: The XP planning process allows the XP "customer" to define the business value of desired features, and uses cost estimates provided by the programmers, to choose what needs to be done and what needs to be deferred. The effect of XP's planning process is that it is easy to steer the project to success.
- There are two planning steps in XP: **The Release planning**: a practice where the customer presents the desired features to the programmers in the team who in return estimate their difficulty**. Iteration planning:** a practice whereby the team is given direction every couple of weeks building a software in 2-weeks “iterations” and delivering running useful software at the end of each iteration.
- **Small Releases**: XP teams put a simple system into production early, and update it frequently on a very short cycle. The team releases running, tested software, delivering business value chosen by the Customer, every iteration. The most important aspect is that the software is visible, and given to the customer, at the end of every iteration. This keeps everything open and tangible.
- **Testing, also known as customer tests:** XP teams focus on validation of the software at all times. Programmers develop software by writing tests first, then software that fulfils the requirements reflected in the tests. Customers provide acceptance tests that enable them to be certain that the features they need are provided. The best way for success is that once the test runs, the team keeps it running correctly thereafter.

- **Metaphor:** XP teams develop a common vision of how the program works, which we call metaphor. In other words they use a common "system of names" and a common system description that guides development and communication.
- **Simple Design**: A program built with XP should be the simplest program that meets the current requirements. There is not much building "for the future". Instead, the focus is on providing business value. Of course it is necessary to ensure that you have a good design. There are design steps in release planning and iteration planning, plus teams engage in quick design sessions and design revisions through refactoring, through the course of the entire project. In an incremental, iterative process like Extreme Programming, good design is then essential. That's why there is so much focus on design throughout the course of the entire development.
- **Refactoring, or design improvement**: Extreme Programming focuses on delivering business value in every iteration. To accomplish this over the course of the whole project, the software must be well designed. This is done by keeping the software clean: without duplication, with high communication, simple, yet complete. Refactoring is, of course, strongly supported by comprehensive testing to be sure that as the design evolves. Thus the customer tests are a critical enabling factor. The XP practices support each other: they are stronger together than separately.
- **Pair Programming**: XP programmers write all production code in pairs, two programmers working together at one machine. This practice ensures that all production code is reviewed by at least one other programmer, and results in better design, better testing, and better code. In fact, many experiments have shown that pair programming produces better software at similar or lower cost than programmers working alone.

- **Collective Code Ownership**: All the code belongs to all the programmers. This lets the team go at full speed, because when something needs changing, it can be changed without delay, which increases code quality and reduces defects.
- **Continuous Integration:** XP teams integrate and build the software system multiple times per day. This keeps all the programmers on the same page, and enables very rapid progress. Perhaps surprisingly, integrating more frequently tends to eliminate integration problems that plague teams who integrate less often
- **Sustainable Pace sometimes known as 40-hour week:** XP programmers work hard and at a pace that can be sustained indefinitely. This means they do not work overtime, unless it's effective, keeping themselves fresh, healthy, as to reduce as much as possible mistakes.
- **On-site Customer:** An XP project is steered by a dedicated individual who is empowered to determine requirements, set priorities, and answer questions as the programmers have them. The effect of being there is that communication improves, with less hard-copy documentation - often one of the most expensive parts of a software project.
- **Coding Standard:** For a team to work effectively in pairs, and to share ownership of all the code, all programmers need to write the code in the same way, with rules that make sure the code communicates clearly.