1.1INTRODUCTION TO SOFTWARE ENGINEERING

The economies of all developed nations are dependent on software. More and more systems are software controlled. Software engineering is concerned with theories, methods and tools for professional software development.

What is software?

Software is set of Computer programs associated with documentation & configuration data that is needed to make these programs operate correctly. A software system consists of a number of programs, configuration files (used to set up programs), system documentation (describes the structure of the system) and user documentation (explains how to use system).Software products may be developed for a particular customer or may be developed for a general market. Software products may be

- Generic - developed to be sold to a range of different customers
- Bespoke (custom) - developed for a single customer according to their specification

What is software engineering?

Ø Software engineering is an engineering discipline which is concerned with all aspects of software production.

Ø Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

What is the difference between software engineering and computer science?

Ø Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software

Ø Computer science theories are currently insufficient to act as a complete underpinning for software engineering

What is the difference between software engineering and system engineering?

Ø System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process

Ø System engineers are involved in system specification, architectural design, integration and deployment

What is a software process?

A set of activities whose goal is the development or evolution of software Generic activities
in all software processes are:

• Specification - what the system should do and its development constraints
• Development - production of the software system
• Validation - checking that the software is what the customer wants
• Evolution - changing the software in response to changing demands

Objectives of Software Engineering:

1.Maintainability –

It should be feasible for the software to evolve to meet changing requirements.

2.Correctness –

A software product is correct, if the different requirements as specified in the SRS document have been correctly implemented.

3.Reusability –

A software product has good reusability, if the different modules of the product can easily be reused to develop new products.

4.Testability –

Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.

5.Reliability –

It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.

6.Portability –

In this case, software can be transferred from one computer system or environment to another.

7.Adaptability –

In this case, software allows differing system constraints and user needs to be satisfied by making changes to the software.

1.2 SOFTWARE DEVELOPMENT LIFE CYCLE

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

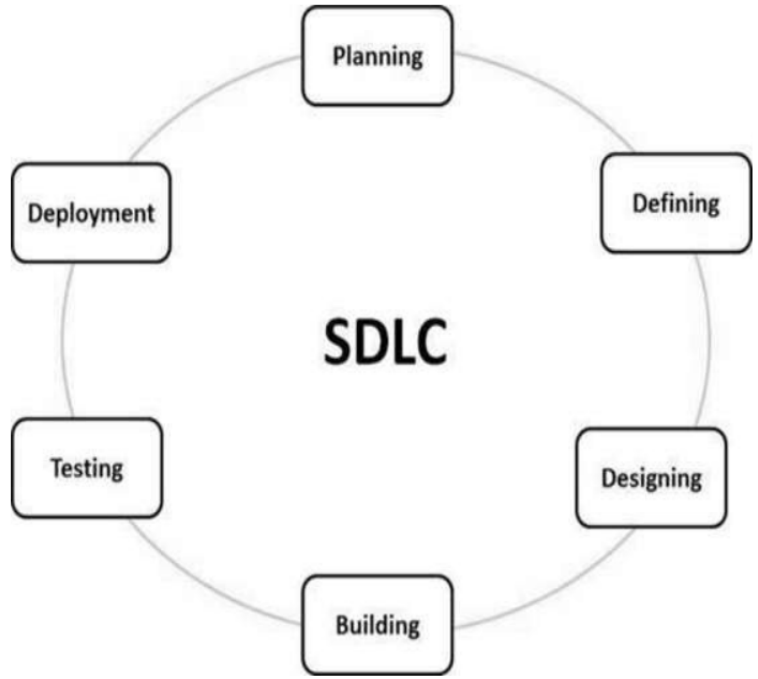
- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC

Stage 1: Planning and Requirement Analysis

- Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.
- Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.



Stage 2: Defining Requirements

- Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

- SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

Stage 4: Building or Developing the Product

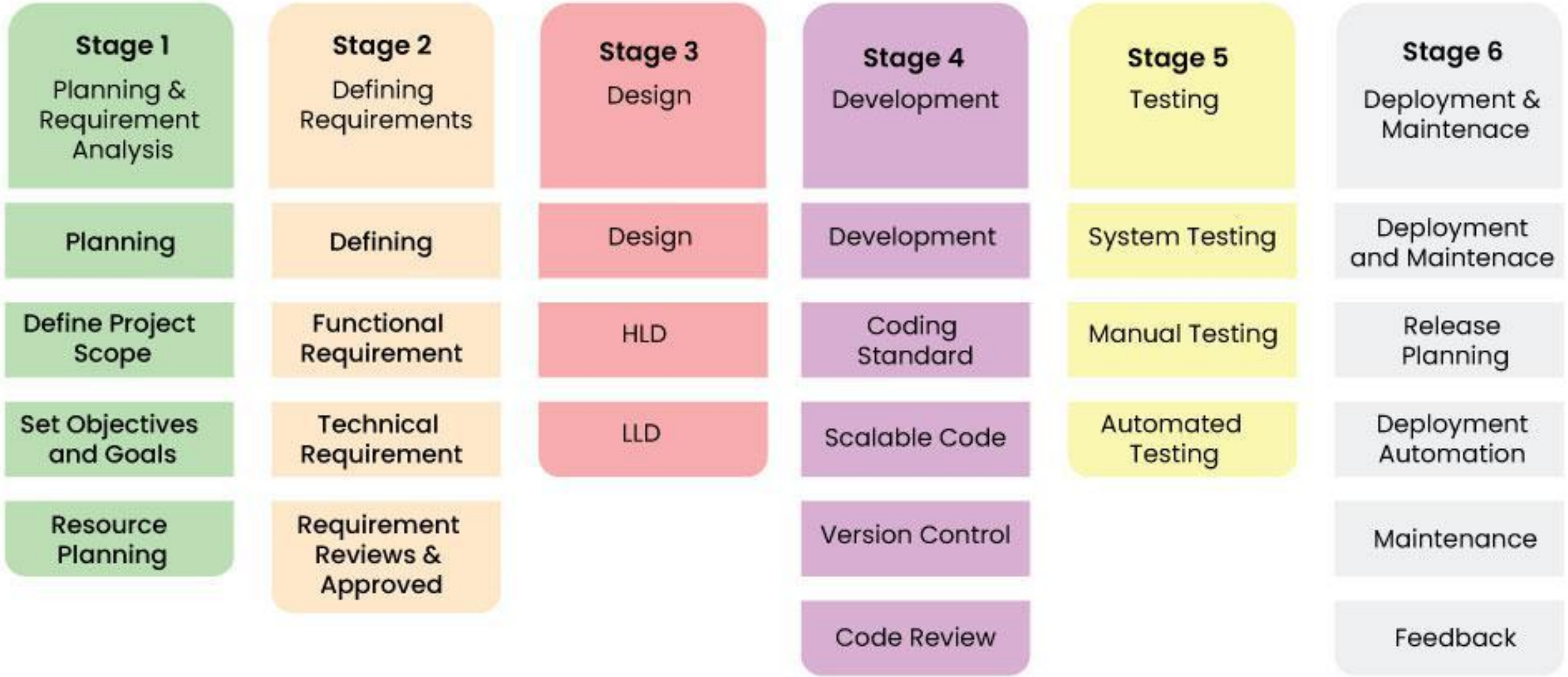
- In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.
- Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed

Stage 5: Testing the Product

- This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

- Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).



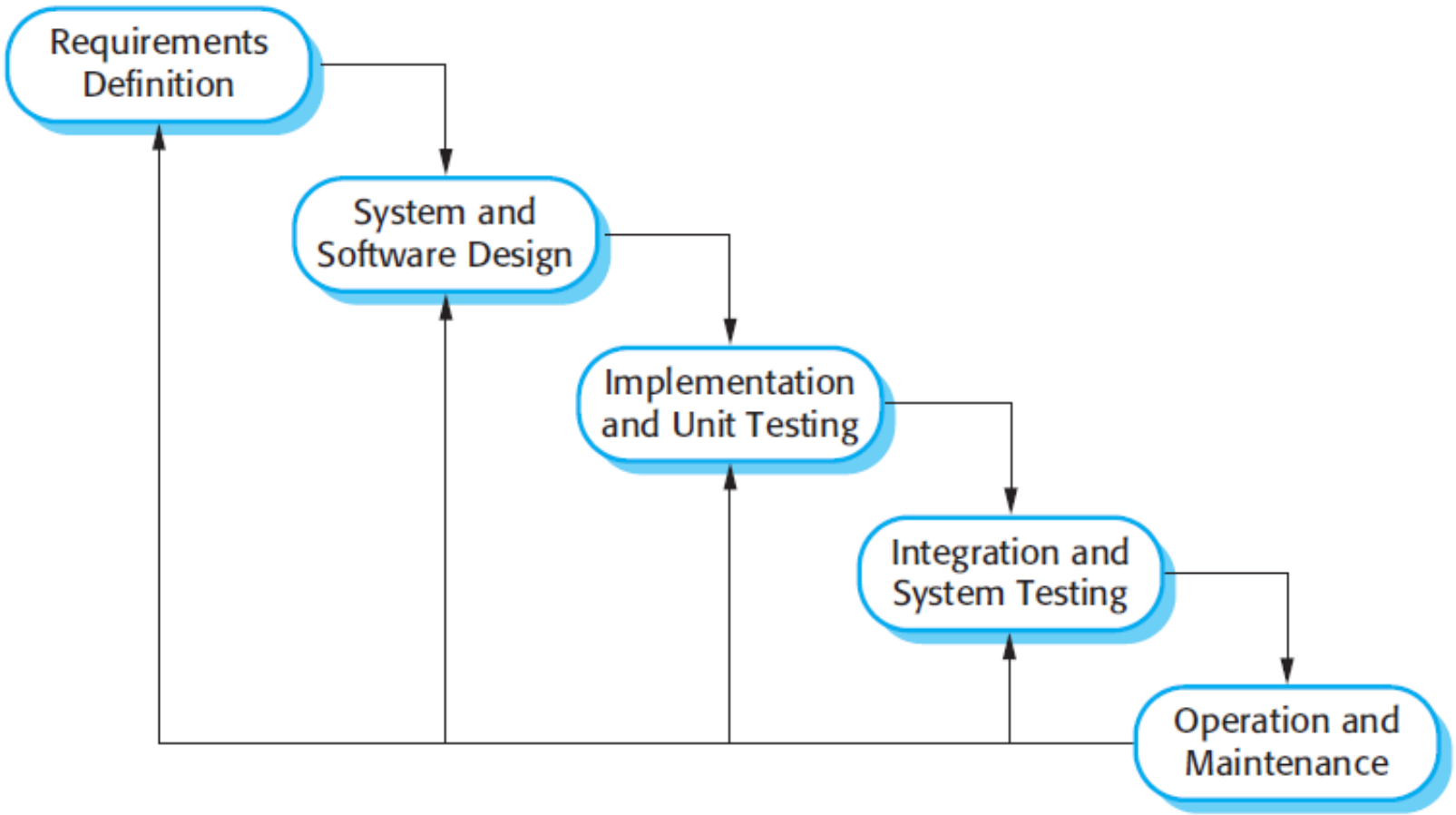
6 Stages of Software Development Life Cycle

THE SOFTWARE PROCESS MODEL

- A software process model is a specified definition of a software process, which is presented from a particular perspective.
- Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described.
- Process models may contain activities, which are part of the software process, software product, and the roles of people involved in software engineering.
- a software process model is a simplified representation of a software process.
- Each process model represents a process from a particular perspective, and thus provides only partial information about that process.
- For example, a process activity model shows the activities and their sequence but may not show the roles of the people involved in these activities.

WATERFALL MODEL

- The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.
- In the waterfall model, you must plan and schedule all of the activities before starting working on them (plan-driven process).
- Plan-driven process is a process where all the activities are planned first, and the progress is measured against the plan.
- The phases of the waterfall model are:
    - Requirements
    - Design
    - Implementation
    - Testing, and
    - Maintenance.


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

The principal stages of the waterfall model directly reflect the fundamental development activities:

1. **Requirements analysis and definition** The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.
2. **System and software design** The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
3. **Implementation and unit testing** During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.
4. **Integration and system testing** The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
5. **Operation and maintenance** Normally (although not necessarily), this is the longest life cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

- The Waterfall model, despite its limitations, has been widely used in real-world software development projects, especially in industries where regulatory compliance and documentation are critical, such as aerospace, defense, and healthcare.
- The waterfall model is consistent with other engineering process models and documentation is produced at each phase.
- This makes the process visible so managers can monitor progress against the development plan.
- **Its major problem** is the inflexible partitioning of the project into distinct stages.
- Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
- In principle, the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.
- However, the waterfall model reflects the type of process used in other engineering projects.
- As is easier to use a common management model for the whole project, software processes based on the waterfall model are still commonly used.
- An important variant of the waterfall model is formal system development, where a mathematical model of a system specification is created.

Advantages	Disadvantages
• Easy to explain to the users. •Structures approach. •Stages and activities are well defined. •Helps to plan and schedule the project. •Verification at each stage ensures early detection of errors/ misunderstanding. •Each phase has specific deliverables.	•No feedback •Very difficult to go back to any stage after it finished. •A little flexibility and adjusting scope is difficult and expensive. •No parallelism •High risk •

SPIRAL MODEL (SDM)

- It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts.
- This model of development combines the features of the prototyping model and the waterfall model.
- The spiral model is favored for large, expensive, and complicated projects.
- This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.

1. Planning

The first phase of the Spiral Model is the planning phase, where the scope of the project is determined and a plan is created for the next iteration of the spiral.

2. Risk Analysis

In the risk analysis phase, the risks associated with the project are identified and evaluated.

3. Engineering

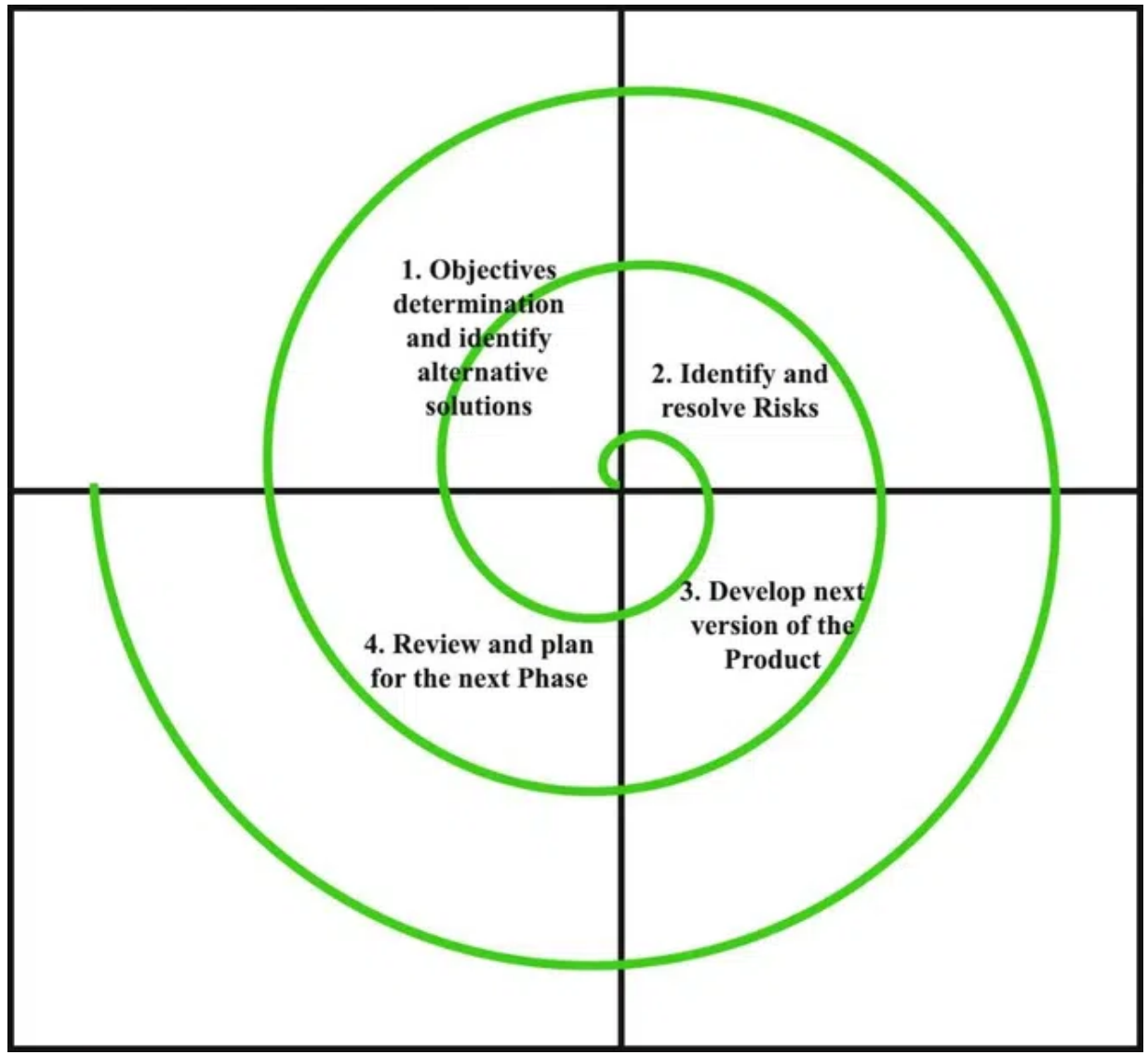
In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.

4. Evaluation

In the evaluation phase, the software is evaluated to determine if it meets the customer’s requirements and if it is of high quality.

5. Planning

The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.


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

Each loop in the spiral is split into four sectors:

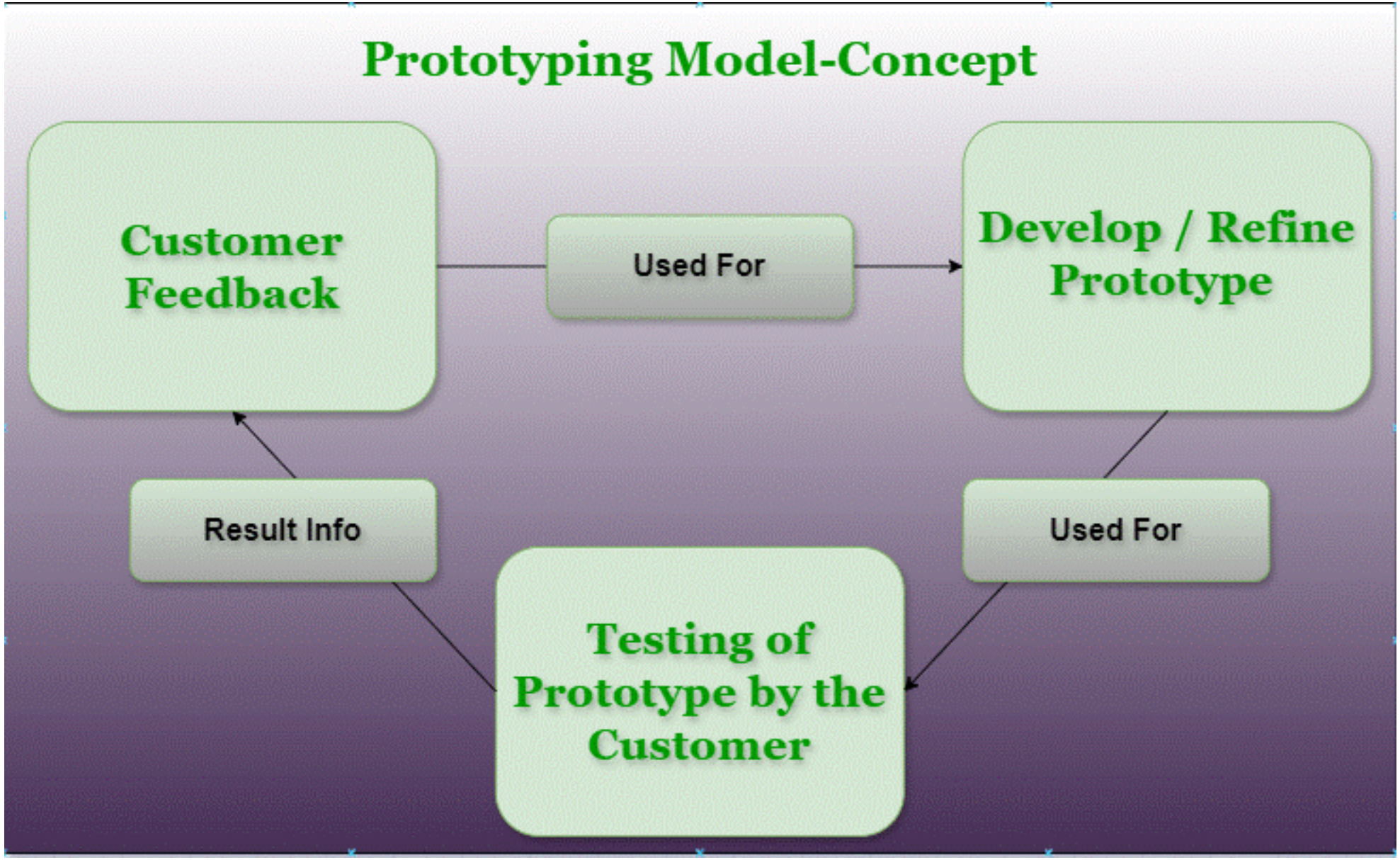
1. **Objectives determination and identify alternative solutions**: Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and 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 are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop the 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.

Advantages	Disadvantages
•Risk Handling •Good for large projects •Flexibility in Requirements •Customer Satisfaction •Improved Communication •Improved Quality	•Complex •Expensive •Too much dependability on Risk Analysis •Difficulty in time managemen •

PROTOTYPING MODEL

1. This model is used when the customers do not know the exact project requirements beforehand.
2. 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.
3. In this process model, the system is partially implemented before or during the analysis phase thereby allowing the customers to see the product early in the life cycle.
4. The process starts by interviewing the customers and developing the incomplete high-level paper model.
5. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer.
6. Once the customer figures out the problems, the prototype is further refined to eliminate them.
7. The process continues until the user approves the prototype and finds the working model to be satisfactory.

1. **Steps of Prototyping Model**
2. Step 1: Requirement Gathering and Analysis: This is the initial step in designing a prototype model. In this phase, users are asked about what they expect or what they want from the system.
3. Step 2: Quick Design: This is the second step in the Prototyping Model. This model covers the basic design of the requirement through which a quick overview can be easily described.
4. Step 3: Build a Prototype: This step helps in building an actual prototype from the knowledge gained from prototype design.
5. Step 4: Initial User Evaluation: This step describes the preliminary testing where the investigation of the performance model occurs, as the customer will tell the strengths and weaknesses of the design, which was sent to the developer.
6. Step 5: Refining Prototype: If any feedback is given by the user, then improving the client's response to feedback and suggestions, the final system is approved.
7. Step 6: Implement Product and Maintain: This is the final step in the phase of the Prototyping Model where the final system is tested and distributed to production, here the program is run regularly to prevent failures.


Prototyping Model-Concept
Customer Feedback
Used For
Develop / Refine Prototype
Result Info
Used For
Testing of Prototype by the Customer

Advantages of Prototyping Model

-
- 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.
- Early feedback from customers and stakeholders can help guide the development process and ensure that the final product meets their needs and expectations.

Disadvantages of the Prototyping Model

- Costly concerning time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
- After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.

Iterative and Incremental Model

- The Iterative and Incremental model is divided into two parts with their respective names. In the incremental model, the process is segregated into small portions called increments where every portion is created based on the previous version.
- This approach ensures that the newer version is better and improved than the previous one. On the other hand, the iterative model makes use of iterations that are activities repeated systematically where a new version is built after every cycle till the desired program is built.
- 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.
- When discussing the iterative method, the concept of incremental development will also often be used liberally and interchangeably, which describes the incremental alterations made during the design and implementation of each new iteration.

- Unlike the more traditional waterfall model, which focuses on a stringent step-by-step process of development stages, the iterative model is best thought of as a cyclical process.
- After an initial planning phase, a small handful of stages are repeated over and over, with each completion of the cycle incrementally improving and iterating on the software.
- Enhancements can quickly be recognized and implemented throughout each iteration, allowing the next iteration to be at least marginally better than the last.
    - Planning & Requirements: As with most any development project, the first step is go through an initial planning stage to map out the specification documents, establish software or hardware requirements, and generally prepare for the upcoming stages of the cycle.
    - Analysis & Design: Once planning is complete, an analysis is performed to nail down the appropriate business logic, database models, and the like that will be required at this stage in the project. The design stage also occurs here, establishing any technical requirements (languages, data layers, services, etc) that will be utilized in order to meet the needs of the analysis stage.
    - Implementation: With the planning and analysis out of the way, the actual implementation and coding process can now begin. All planning, specification, and design docs up to this point are coded and implemented into this initial iteration of the project.

•**Testing**: Once this current build iteration has been coded and implemented, the next step is to go through a series of testing procedures to identify and locate any potential bugs or issues that have cropped up.
•**Evaluation**: Once all prior stages have been completed, it is time for a thorough evaluation of development up to this stage. This allows the entire team, as well as clients or other outside parties, to examine where the project is at, where it needs to be, what can or should change, and so on.

Example : LMS

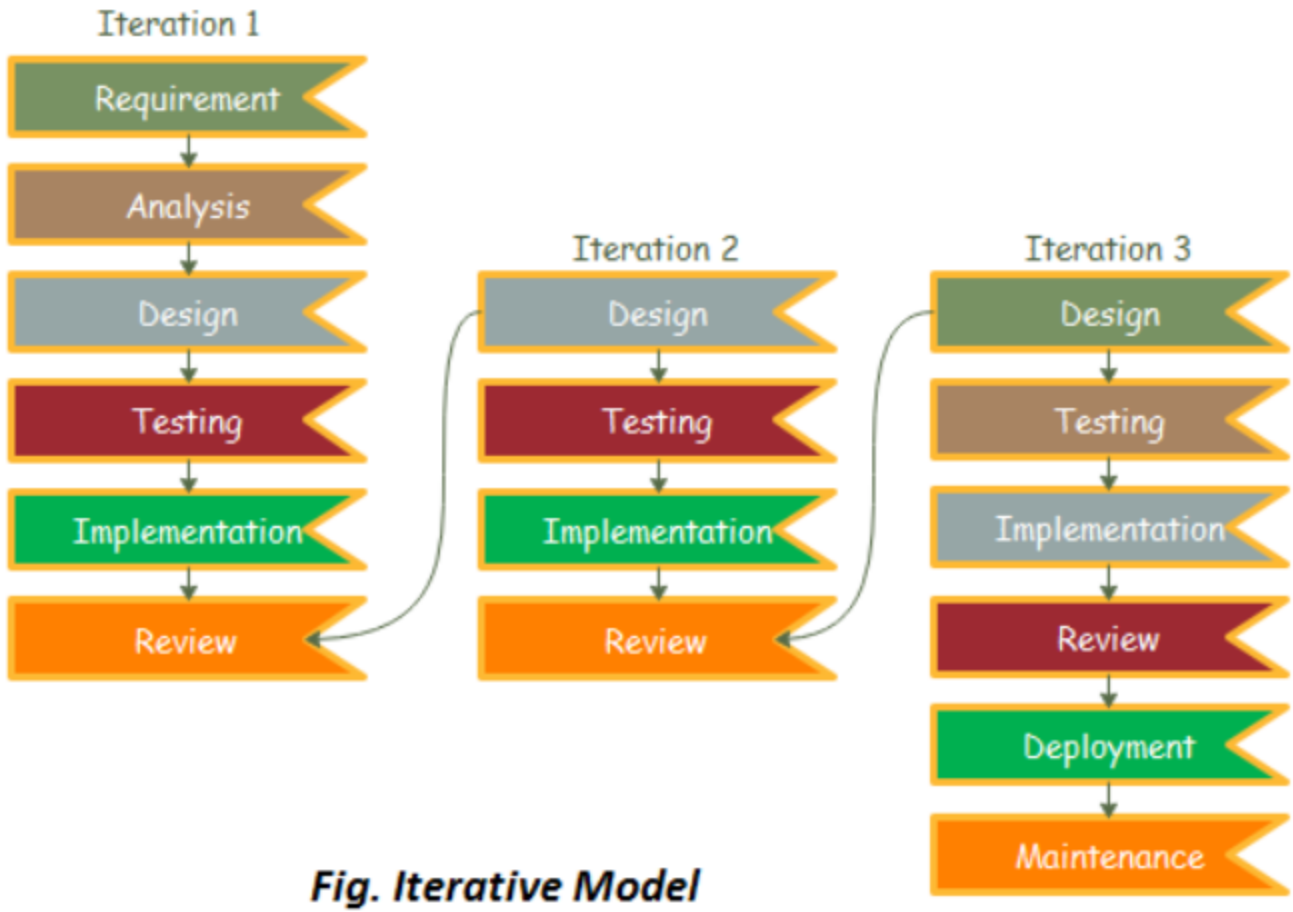


Fig. Iterative Model

Advantages of the Iterative Model

The benefits of this software development model include reduction in delivery cost, faster software delivery, flexibility to make changes in the software midway, and the ability to create organized prerequisites.

Module to module working

Disadvantages of the Iterative Model

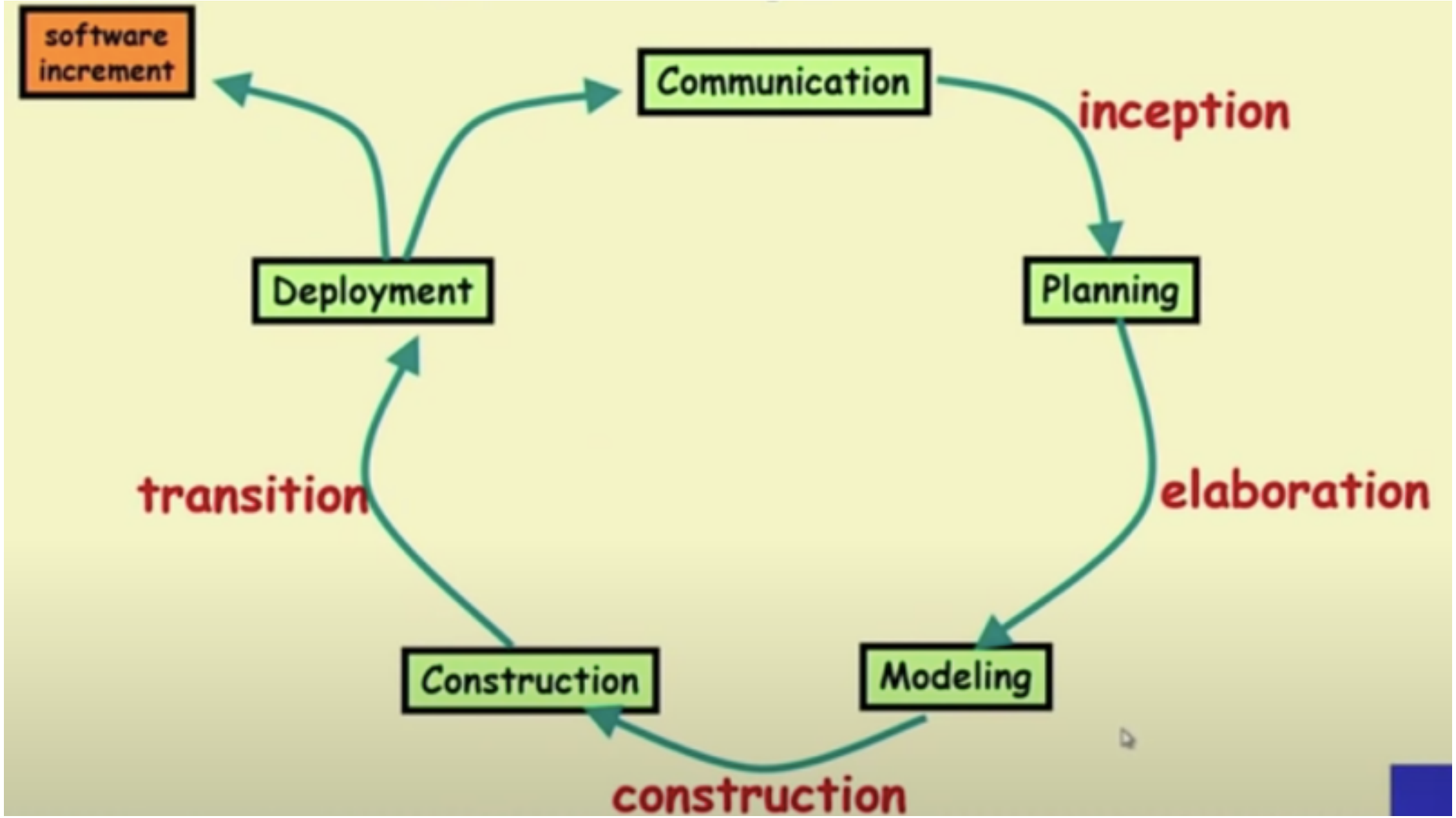
On the other hand, there are several drawbacks to this model as well. The first one is the actual cost of the program. Even though it may reduce the delivery cost, the overall cost is still on the higher side. In addition to that, it also needs the arrangement of cycles and requires a robust plan to make any alterations to the plan in case of any necessary changes.

Unified Process

Unified Process is based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation. The system is developed incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental software development.

Unified Process Phases

- **Inception Phas**e: Communication with the customer is made, and all the features are identified and plan for the development process.
- **Elaboration Phas**e: Identified features are modeled.
- **Construction Phase**: The is constructed and developed.
- **Transition Phase**: The software is deployed on the client-side.


software increment
Communication
inception
Deployment
Planning
transition
elaboration
Construction
Modeling
construction

- **Inception**: The inception phase of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.
- **Elaboration**:Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software—the use case model, the requirements model, the design model, the implementation model, and the deployment model.
- **Construction**: The construction phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users. To accomplish this, requirements and design models that were started during the elaboration phase are completed to reflect the final version of the software increment.
- **Transition**: The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. y. Software is given to end users for beta testing and user feedback reports both defects and necessary changes. In addition, the software team creates the necessary support information that is required for the release. At the conclusion of the transition phase, the software increment becomes a usable software release.

Agile software development

- In the 1980s and early 1990s, there was a widespread view that the best way to achieve better software was through careful project planning, formalized quality assurance, the use of analysis and design methods supported by CASE tools, and controlled and rigorous software development processes.
- This view came from the software engineering community that was responsible for developing large, long lived software systems such as aerospace and government systems.
- This software was developed by large teams working for different companies.
- Teams were often geographically dispersed and worked on the software for long periods of time.
- An example of this type of software is the control systems for a modern aircraft, which might take up to 10 years from initial specification to deployment.
- These plan driven approaches involve a significant overhead in planning, designing, and documenting the system.
- This overhead is justified when the work of multiple development teams has to be coordinated, when the system is a critical system, and when many different people will be involved in maintaining the software over its lifetime.

Agile is a type of software development methodology that anticipates the need for flexibility and applies a level of pragmatism to the delivery of the finished product. Agile software development requires a cultural shift in many companies because it focuses on the clean delivery of individual pieces or parts of the software and not on the entire application.

The collaborative culture facilitated by Agile also improves efficiency throughout the organization as teams work together and understand their specific roles in the process. Finally, companies using Agile software development can feel confident that they are releasing a high-quality product because testing is performed throughout development.

This provides the opportunity to make changes as needed and alert teams to any potential issues.

The 12 principles of Agile

The Agile Manifesto also outlined 12 core principles for the development process. They are as follows:

1. Satisfy customers through early and continuous delivery of valuable work.
2. Break big work down into smaller tasks that can be completed quickly.
3. Recognize that the best work emerges from self-organised teams.
4. Provide motivated individuals with the environment and support they need and trust them to get the job done.
5. Create processes that promote sustainable efforts.
6. Maintain a constant pace for completed work.
7. Welcome changing requirements, even late in a project.
8. Assemble the project team and business owners on a daily basis throughout the project.
9. Have the team reflect at regular intervals on how to become more effective, then tune and adjust behaviour accordingly.
10. Measure progress by the amount of completed work.
11. Continually seek excellence.
12. Harness change for a competitive advantage.

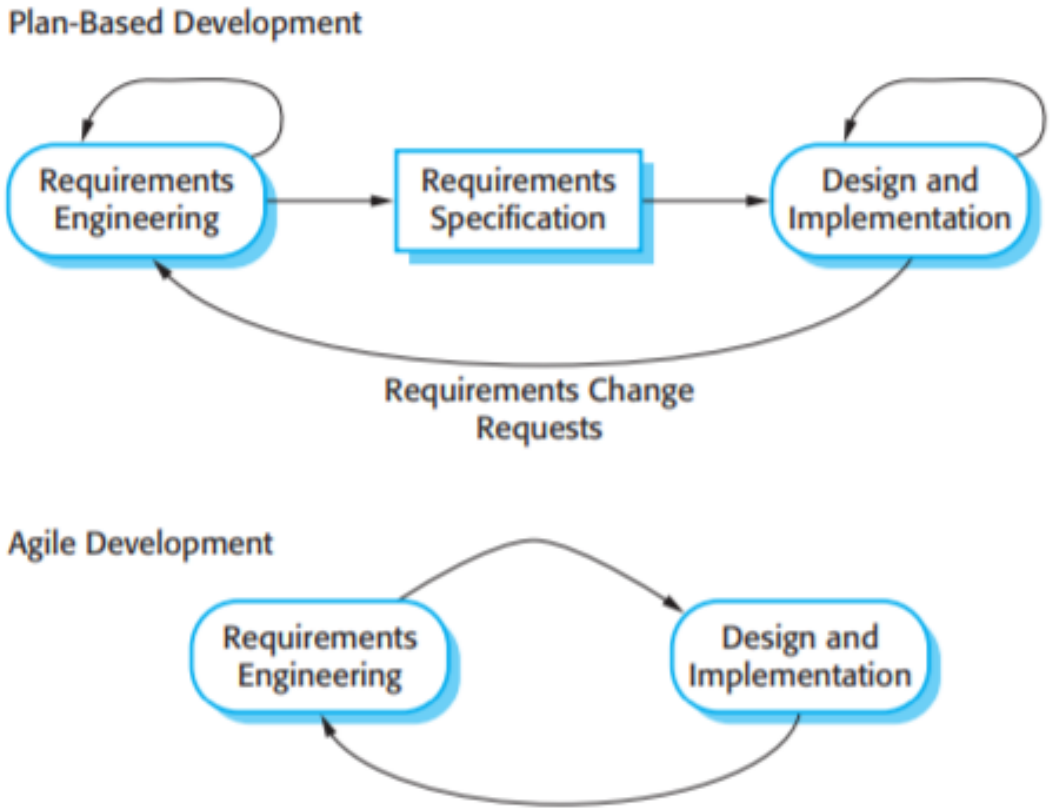


Figure Plan-driven and agile specification

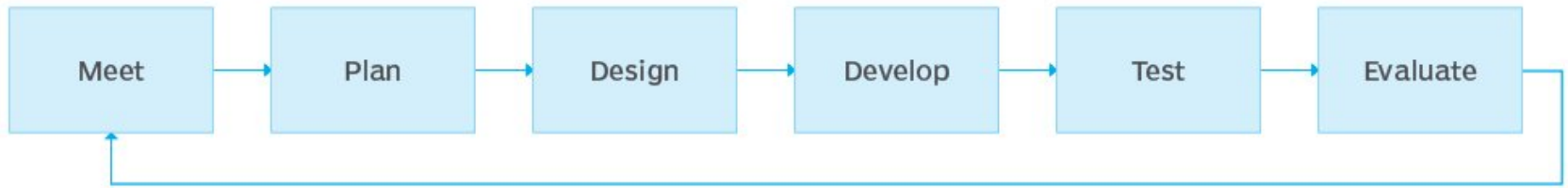
The Agile software development cycle

The Agile software development cycle can be broken down into the following six steps:

- concept
- inception
- iteration/construction
- release
- production
- retirement

- The first step, ***concept***, involves the identification of business opportunities in each potential project as well as an estimation of the time and work that will be required to complete the project.
- During the second step, ***inception***, team members are identified, funding is established and the initial requirements are discussed with the customer. A timeline should also be created that outlines the various responsibilities of teams and clearly defines when work is expected to be completed for each sprint. A sprint is a set period of time during which specific work has to be completed and made ready for review.
- The third step, ***iteration/construction***, is when teams start creating working software based on requirements and continuous feedback. The Agile software development cycle relies on iterations -- or single development cycles -- that build upon each other and lead into the next step of the overall development process until the project is completed. Each iteration typically lasts between two to four weeks, with a set completion date.
- The fourth step, ***release***, involves final QA testing, resolution of any remaining defects, finalization of the system and user documentation and, at the end, release of the final iteration into production.
- After the release, the fifth step, ***production***, focuses on the ongoing support necessary to maintain the software. The development teams must keep the software running smoothly while also teaching users exactly how to use it. The production phase continues until the support has ended or the product is planned for retirement.
- The final step, ***retirement***, incorporates all end-of-life activities, such as notifying customers and final migration. The system release must be removed from production. This is usually done when a system needs to be replaced by a new release or if the system becomes outdated.

Agile software development cycle



Agile Testing Methods

After looking at the different phases of the agile model, this section describes the different testing methods involved in this model. We will be discussing each and every method in detail.

Scrum

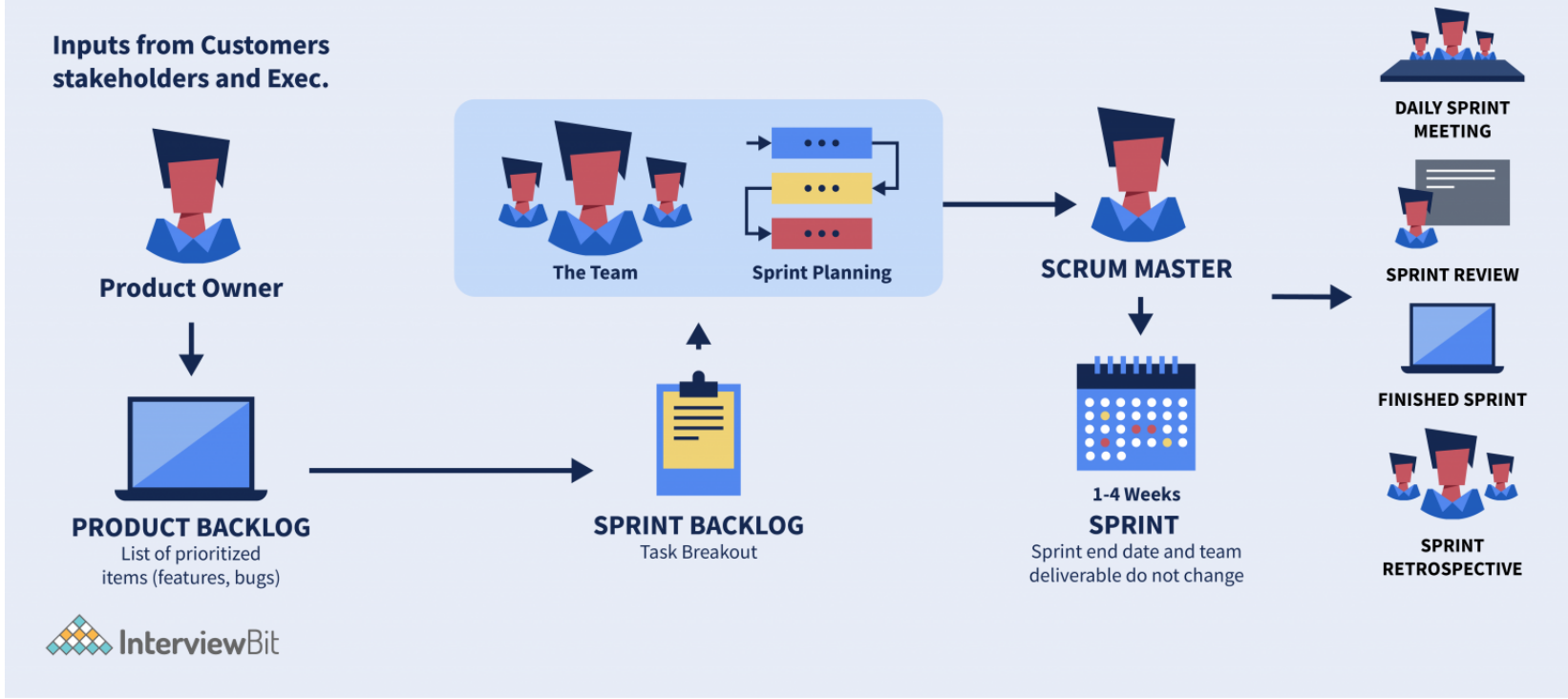
Scrum is formed from the activity that takes place during a rugby match. It is an agile development methodology that focuses on task management within a team-based development environment. It promotes working in small teams and believes in empowering the development team. There are three positions in it, each with its own set of responsibilities described below:

- **Scrum Master:** Sets up the team, holds sprint meetings, and removes roadblocks to development.
- **Product Owner:** Creates the product backlog, prioritizes the delay, and is in charge of the functionality distribution on each repeat.
- **Scrum Team:** The team organizes and oversees their work in order to accomplish the sprint or cycle.

Apart from this, the below-mentioned points give a glimpse of a process flow that is followed in a scrum:

- Each iteration of a scrum is termed as Sprint.
- A product backlog is a list that contains all of the information needed to create the final product.
- The top user stories from the Product backlog are chosen and translated into Sprint backlogs during each Sprint.
- The team works on the sprint backlog that has been established.
- The team double-checks the everyday job.
- The team delivers product functionality at the end of the sprint.

SCRUM SOFTWARE DEVELOPMENT PROCESS



Extreme Programming (XP)

When clients' needs or specifications are continually changing, or when they are unsure about the system's functionality, the Extreme Programming technique comes in handy. It encourages numerous "releases" of the product in short development cycles, which increases the system's efficiency and provides a checkpoint where any client requirements may be quickly incorporated. The XP creates software with the client in mind. The following are some of the beneficial practices identified in the extreme programming model and proposed for maximizing their use:

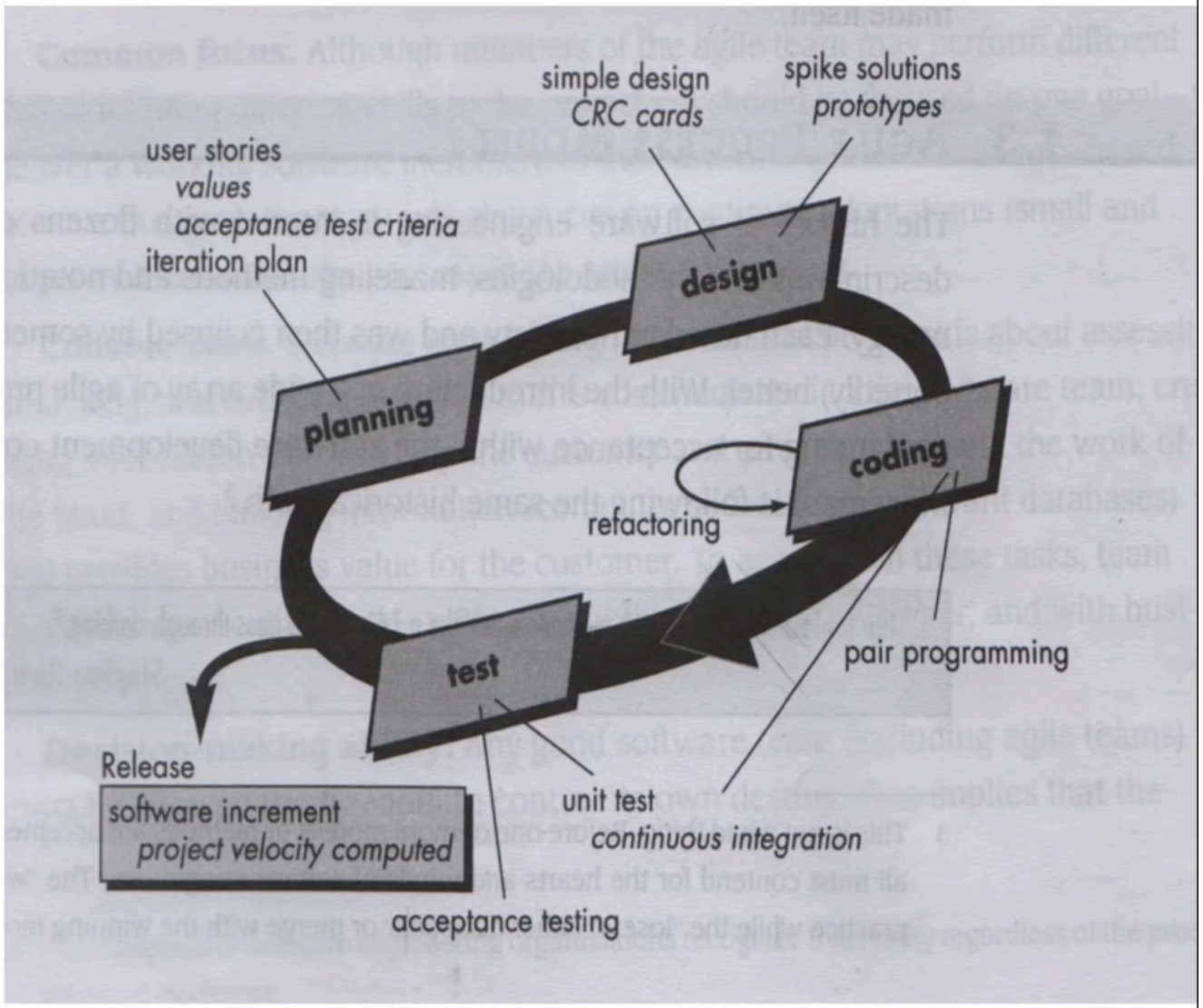
- Code Review: Code review effectively finds and corrects mistakes. It describes pair programming as the coding and review of written code carried out by a pair of programmers who alternate their work every hour.
- Testing: Testing code aids in the removal of faults and increases its dependability. To continuously write and execute test cases, XP recommends test-driven development (TDD). Test cases are written before any code is written in the TDD approach.
- Incremental Development is beneficial since client feedback is obtained, and the development team uses this information to create new increments every few days following each iteration.
- Simplicity: facilitates the development of high-quality code as well as its testing and debugging.
- Design: To produce high-quality software, it's critical to have a decent design. As a result, everyone should design on a daily basis.
- Integration Testing: It aids in the detection of flaws at the interfaces of various functionalities. According to extreme programming, developers should build and test integration numerous times a day to achieve continuous integration.

• EXTREME PROGRAMMING(XP)

- XP is a lightweight , efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.
- Small to medium sized teams that work under vague and rapidly changing requirements.
- The five values of XP are communication, simplicity, feedback, courage, and respect
- Follows Object Oriented approach.

Spike solution:A **spike solution** is a very simple program to explore potential **solutions.** Difficult designs should be modeled using prototype.

Refactoring: Improves the internal structure of the code but external behavior not affected.



- Class-Responsibility Collaborator(CRC).

Class: *Librarian*	
Responsibilities	**Collaborators**
check in book	*Book*
check out book	*Book, Borrower*
search for book	*Book*
knows all books	
search for borrower	*Borrower*
knows all borrowers	

Class CardReader	
Responsibilities	Collaborators
Tell ATM when card is inserted	ATM
Read information from card	Card
Eject card	
Retain card	

ACCEPTANCE TESTING:-During the process of manufacturing a ballpoint pen, the cap, the body, the tail and clip, the ink cartridge and the ballpoint are produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. When the complete pen is integrated, System Testing is performed. Once System Testing is complete, Acceptance Testing is performed so as to confirm that the ballpoint pen is ready to be made available to the end-users.

• ADVANTAGES:-

1. Fewer documentation required.
2. Collaboration with customers.
3. Flexibility to developers.
4. Easy to manage.

• DISADVANTAGES:-

1. Depends heavily on customer interaction.
2. Transfer of technology to new team members may be quite challenging due to lack of documentation.

Feature Driven Development (FDD)

The major goal of feature-driven development is to provide clients with timely updated and functional software. At all levels of FDD, reporting and progress tracking is required. The “Designing and Building” features are the center of this method. Below mentioned points are the lifecycle of FDD:

- First of all entire model is built
- After the model is done, the feature list is prepared
- Then plan according to feature
- Design according to feature
- At last build according to feature

Features of FDD

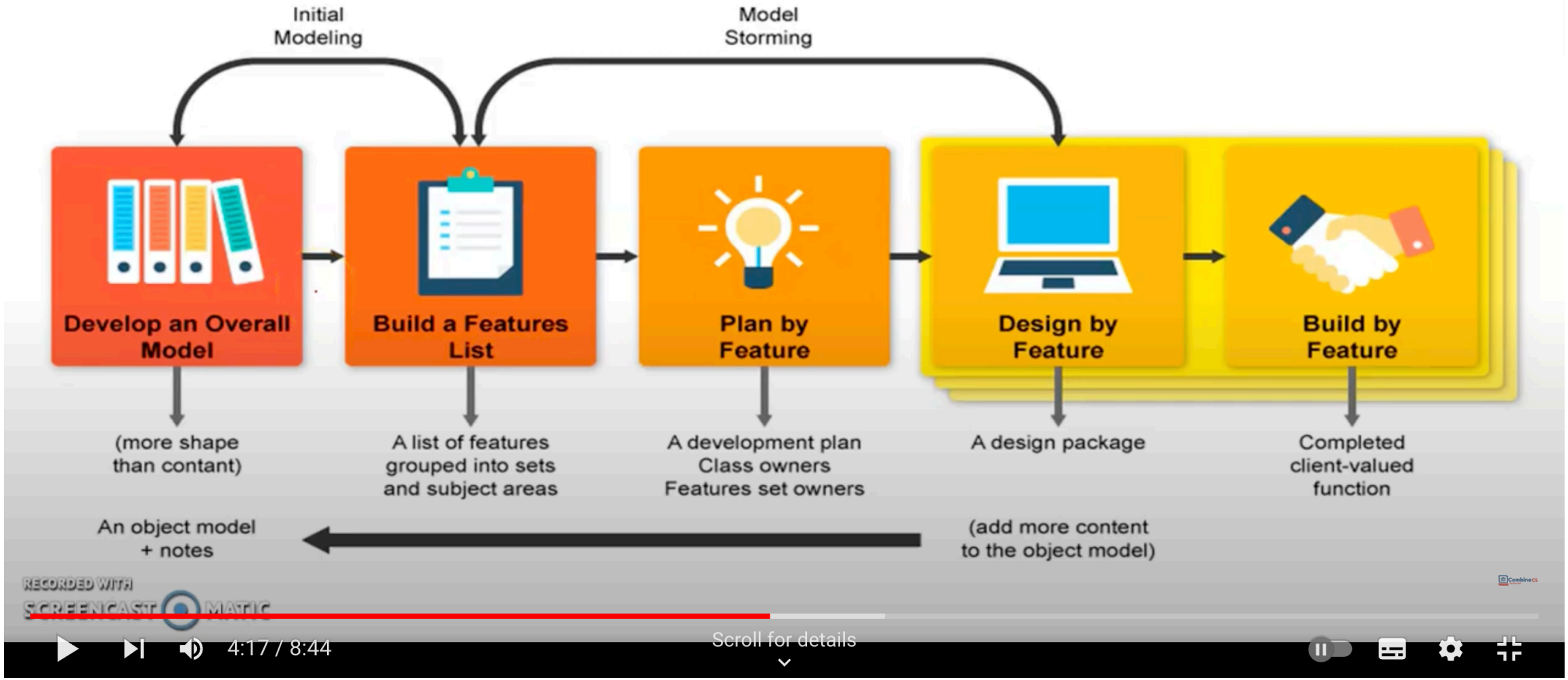
- Customer Focused
- Short Iterations are there i.e. The FDD lifecycle uses basic and short iterations to efficiently complete work on schedule and keeps massive projects moving.
- The domain model and feature list are built in the first two activities of the lifecycle, while the latter two activities get more than 70% of the effort.
- FDD ensures that new features are added to the software on a regular basis, ensuring the project’s long-term success.

- What is considered a feature in Feature Driven Development?
- A feature is a small, client-valued function expressed in the form <action><result><object>.
- Feature-Driven Development (FDD) is a client-centric, architecture-centric, and pragmatic (practical approach) software process. As the name implies, features are an important aspect of FDD.

FDD

- **Example** - "complete the login process" might be considered a feature in the Feature Driven Development (FDD) methodology.

- The resulting domain object model provides an overall framework in which to add features. Developing by Feature. Any function that is too complex to be implemented within two weeks is further decomposed into smaller functions until each sub-problem is small enough to be called a feature.


Initial
Modeling
Model
Storming
Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than contant)
A list of features grouped into sets and subject areas
A development plan
Class owners
Features set owners
A design package
Completed client-valued function
An object model
+ notes
(add more content to the object model)
RECORDED WITH SCREENCAST-O-MATIC
4:17 / 8:44
Scroll for details

FDD has five (5) basic processes steps:

1. **Developing an overall model:** Cross-functional, iterative and highly collaborative. FDD pushes team members to work together to build an object model of the domain area as guided by the Chief Architect. When detailed domain models are created, these models are progressively merged into an overall model after.
2. **Building the list of features:** By using the model on the previous step, the team or the chief programmer builds a list of features that would be useful to users and could be completed along a set timeline for release.
3. **Planning by feature** It's all about organizing. Here, plans are laid in which order the features will be implemented. Teams are then selected and assigned feature sets.
4. **Designing by feature:** On this stage, the chief programmer chooses the features for development and assigns them to feature teams consisting of the project manager; the chief architect; the development manager; the domain expert; the class owner; and the chief programmer.
5. **Building by feature:** Feature teams have complete coding, testing, and documentation of each feature, then advance the feature to the main build.

Lean Software Development

The “Just in Time Production” premise underpins the lean software development process. Its goal is to speed up software development while lowering costs. The process of lean development can be broken down into seven parts.

- Getting Rid of Waste
- Boosting Learning
- Deferring Commitment (deciding as late as possible)
- Delivery in a timely manner
- Empowering the Team
- Building Integrity
- Optimize the entire process.


VOICE OF THE CUSTOMER (VOC)
AWESOME!
Excellent
Very Good
Satisfactory
Marginal
OPERATING SYSTEM (OS)
MANAGEMENT INFRASTRUCTURE (MI)
MINDSET & BEHAVIOR (MB)
I'LL DO MY BEST!
TAKE IT EASY ...
These four dimentions
2:28 / 7:55
Scroll for details

Agile Lean Software Development

Kanban

Kanban is a Japanese word that refers to a card that contains all of the information needed to complete a product at each stage of its journey to completion. This framework or method is widely used in software testing, particularly in Agile methodologies. It is a visual project management model that emphasizes continual improvement and workflow openness. Kanban in software testing is all about showing project status to clearly understand what's waiting to be picked up in a backlog, what work is now in progress, and what work is finished. In software testing, Kanban helps to improve organization, maintain a consistent project flow, and promote transparency on the state of work throughout the process.

What is Kanban? Understanding Kanban an Agile Framework - Agile Digest

Movie Theater.
2 Hours Show

Show Duration : **2 Hours Each.**

Max Seat Capacity : **50**

Show Start and End on **Scheduled date and Time**

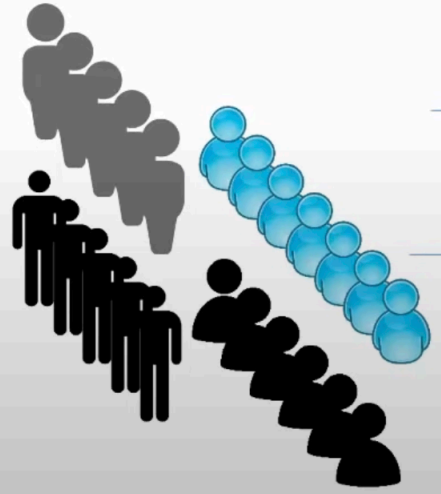
People for Future Show

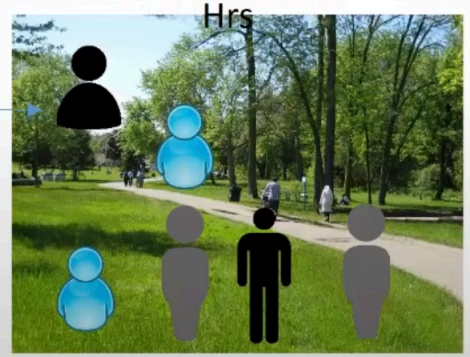
People Watching Current Show

People Recently watched a Show

People watched a Show earlier

Public Park - Open 24 Hrs





Show Duration : **No Defined Show Time**

Max Capacity : **Unlimited**

Start and End on : **No Defined start and End Time**

Park Rule : **Max 6 Persons at a time**

Public Waiting outside to get in the park

6 People inside the Park

People already visited the park

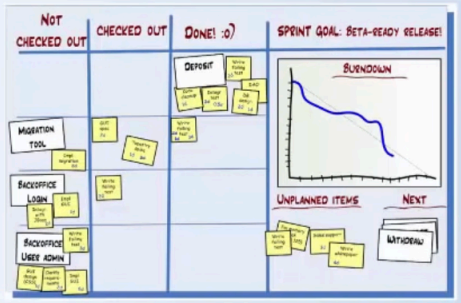
Agile Digest

7:18 / 46:00 • Example of Park ›

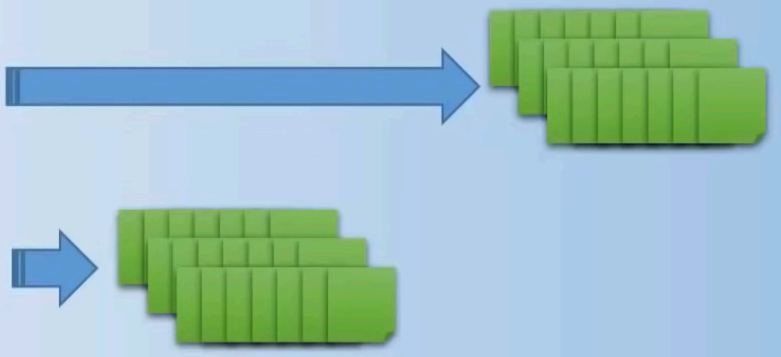
Scroll for details

Backlog, Multiple Stories

Sprint of 2 Weeks



Planned stories are in execution



Past Sprint

Past Sprint

Sprint Duration : **2 Weeks**

Max Capacity : **Team Capacity in Hours**

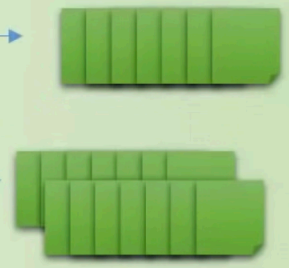
Sprint Start and End on **Scheduled date and Time**

SCRUM

Kan Ban Board

Active Backlog

6 Top priority Story in Progress



Stories Completed

Board Duration : **No Defined Show Time**

Max Capacity : **Unlimited**

Start and End on : **No Defined start and End Time**

Board Rule : **Max 6 stories at a time.**

Kan Ban

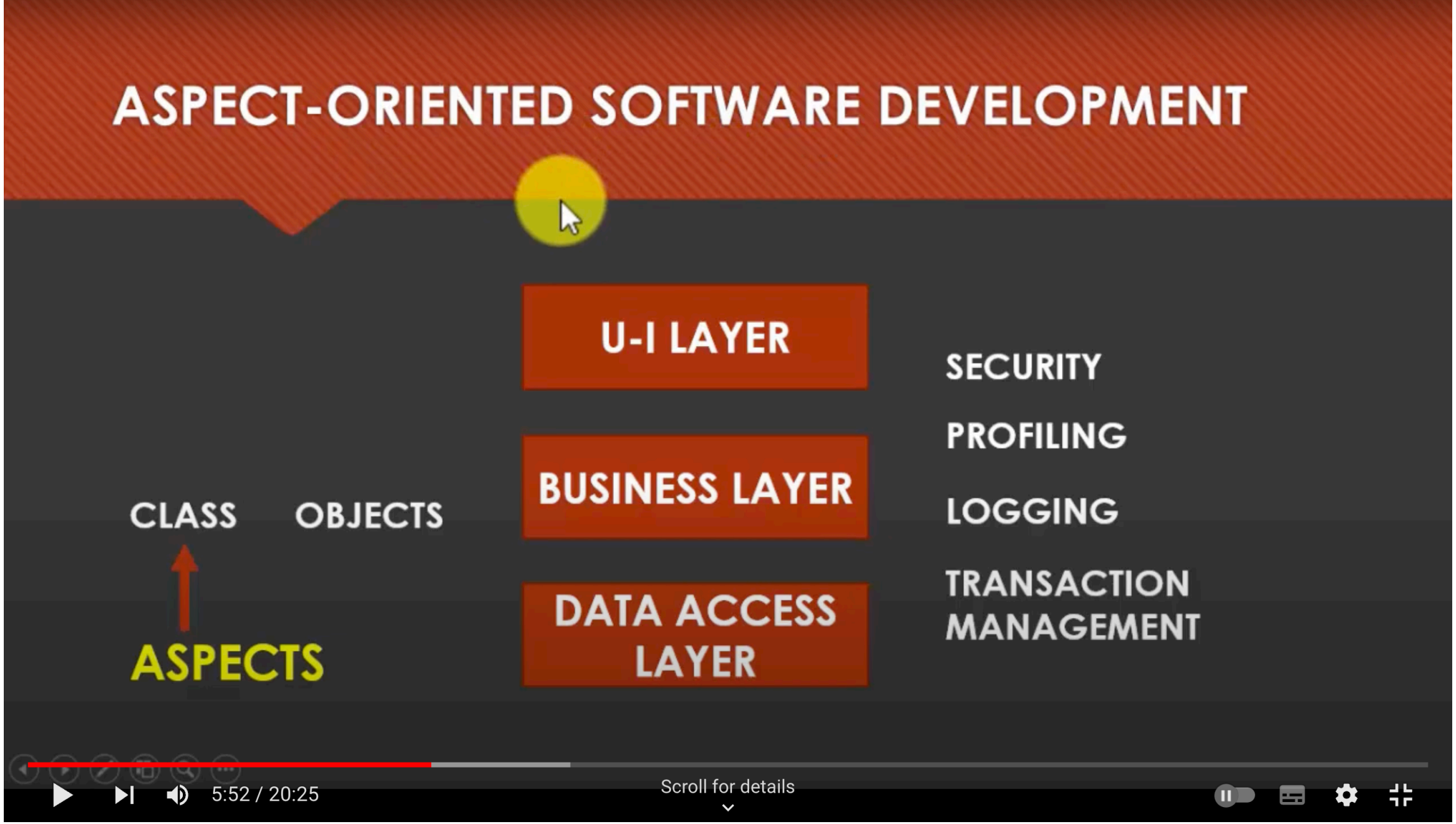
Agile Digest

Agile Digest

Aspect-oriented software engineering and process

Aspect-oriented software engineering (AOSE) is an approach to software development that aims to modularize crosscutting concerns, such as logging, security, and transaction management, that cut across multiple modules of a software system. These concerns often lead to code scattering and tangling, making the system harder to understand, maintain, and evolve.

In AOSE, these crosscutting concerns are captured separately from the main system's concerns and are called aspects. Aspects encapsulate the behavior that affects multiple modules, and they are woven into the system at specific points to provide the desired functionality without modifying the core logic of the modules.


ASPECT-ORIENTED SOFTWARE DEVELOPMENT
U-I LAYER
BUSINESS LAYER
DATA ACCESS LAYER
CLASS
OBJECTS
ASPECTS
SECURITY
PROFILING
LOGGING
TRANSACTION MANAGEMENT
5:52 / 20:25
Scroll for details

AOSD CONCEPTS AND TERMINOLOGY

- **JOIN POINT:**
- A well-defined point in the program's execution where an aspect is invoked.
- **POINTCUT:**
- A specifically defined subset of join points.
- **ADVICE BODY:**
- Code that is executed when a join point is reached.
- **WEAVERS:**
- Specific type of compilers that compose the aspect's implementation with other modules.

1. **Identification of Crosscutting Concerns**: The first step is to identify crosscutting concerns in the system. This involves analyzing the system requirements, design, and code to identify recurring patterns that affect multiple modules.
2. **Aspect Design**: Once crosscutting concerns are identified, aspects are designed to encapsulate the behavior associated with these concerns. This includes defining join points (points in the execution of the system where aspects can be applied), pointcuts (specifications that determine which join points are affected), and advice (the behavior associated with a pointcut).
3. **Aspect Implementation**: After designing aspects, they need to be implemented. This involves writing the code for the advice, which specifies what should be done when a pointcut is reached. Aspects are typically implemented separately from the main system code.
4. **Weaving**: Weaving is the process of integrating aspects into the main system. This can be done at compile-time, load-time, or runtime. During weaving, the aspects are combined with the main system code to produce the final executable.
5. **Testing and Validation**: Once the aspects are woven into the system, testing and validation are performed to ensure that the system behaves as expected. This involves testing both the core functionality of the system and the behavior introduced by the aspects.
6. **Maintenance and Evolution**: As the system evolves, the aspects may need to be updated or modified to accommodate changes. This involves revisiting the identification, design, implementation, and weaving steps as necessary.

ADVANTAGES AND DISADVANTAGES OF AOSD

Advantage	Disadvantage
Increased modularity.	Reduced efficiency due to increased overhead aspects.
Increased maintainability.	Security risks involving aspects.
Increased reusability.	Lack of a formalized process.
Reduction in the size of the code.	Lack of UML support. Lack of systematic methods of testing.