Segment 2: Software Processes

Software Development Process

- ✓ A set of related activities which lead to the production of a high quality software product
- ✓ It is also called Software Development Life Cycle (SDLC)
- ✓ There is no ideal process
- ✓ Different organizations have developed completely different approaches to software development based on their need and capacity.

Why a Methodology is Important for software Engineering?

The oldest software development techniques are of the big bang type and they do not contain any methodology. In the big bang type, a project is initiated and people with adequate skills are hired to do the project work. The management in this scenario has no idea how the project will be executed. They just hope that the people will somehow develop the software product. Since no particular methodology is in place, it is the people who will decide and keep working as per their individual plans. This kind of technique was used during the early days of software development because there were no well-established software engineering methodologies available at that time. In this type of software development, the results are unpredictable. Later, software practitioners realized the need for software engineering methodologies. Some methodologies are discussed in the next few sections.

When you are building a software product, you implicitly acknowledge that you are building it for commercial or critical use and you are using the most competitive, cost-effective, and sound engineering techniques available. In order to apply the engineering techniques to build a product, a methodology that has been proven successful and effective in terms of cost, time, quality, and so on is needed.

Systems Development Life Cycle (SDLC)

The SDLC is the process of determining how an information system (IS) can support business needs, designing the system, building it, and delivering it to users.

In any given SDLC phase, the project can return to an earlier phase, if necessary. In the systems development life cycle, it is also possible to complete some activities in one phase in parallel with some activities of another phase. Sometimes the life cycle is iterative; that is, phases are repeated as required until an acceptable system is found. Some systems analysts consider the life cycle to be a spiral, constantly cycling through the phases at different levels of detail, as illustrated in below figure.

Figure: SDLC with six phases (1) Planning and analysis, (2) Design, (3) Implementation or coding, (4) Testing, (5) Deployment. (6) Maintenance

There are following six phases in every Software development life cycle model:

1) ***Planning and analysis:*** During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

   Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

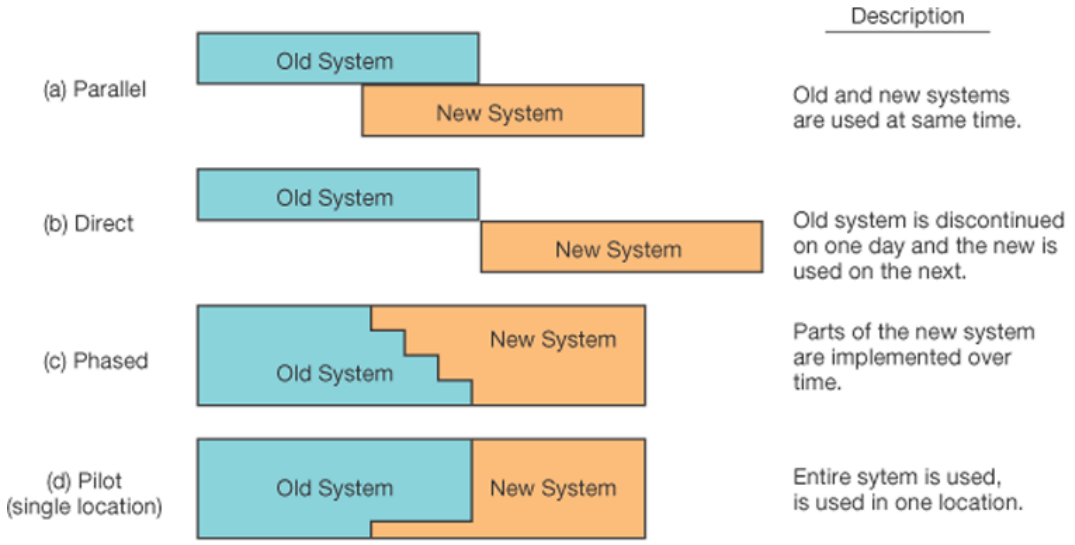
2) ***Design:*** Here, the Project Architect will prepare the system design. He will specify the hardware, system requirement and design the application architecture. There are 2 parts in the system design: high-level design and low-level design. In high-level design, we design the different blocks of the application. In low-level design, we write the pseudo-code.

3) ***Implementation / Coding:*** On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

4) ***Testing:*** After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase all types of functional testing like unit testing, integration testing, system testing, acceptance testing are done as well as non-functional testing are also done.

5) ***Deployment:*** After successful testing the product is delivered / deployed to the customer for their use.

During deployment, there are four widely used strategies i.e., Parallel, Direct, Phased, and Pilot.



6) Maintenance: Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance. Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance :

- *Corrective Maintenance:* This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- *Perfective Maintenance:* This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
- *Adaptive Maintenance:* Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

Joint Application Design (JAD)

- Joint application development (JAD) is a structured process in which users, managers, and analysts work together for several days in a series of intensive meetings to specify or review system requirements.
- It was developed by IBM in the late 1970s.
- The following is a list of typical JAD participants:

JAD participant	Role
JAD project leader	Develops an agenda, acts as a facilitator, and leads the JAD session
Top management	Provides enterprise-level authorization and support for the project
Managers	Provide department-level support for the project and understanding of how the project must support business functions and requirements
Users	Provide operational-level input on current operations, desired changes, input and output requirements, user interface issues, and how the project will support day-to-day tasks
Systems analysts and other IT staff members	Provide technical assistance and resources for JAD team members on issues such as security, backup, hardware, software, and network capability
Recorder	Documents results of JAD sessions and works with systems analysts to build system models and develop CASE tool documentation

Figure: A typical room layout for a JAD session

JAD Process Steps

- ***Define Session:*** Define the purpose, scope, and objectives of the JAD session, selecting the JAD team, invite and obtain commitment to attend sessions from the appropriate stakeholders, and schedule the session. It is important to obtain management commitment to support the process and identify the appropriate stakeholders.
- ***Research Product:*** Become more familiar with the product or service, gather preliminary information, obtaining any models.

- ***Prepare:*** Prepare any visual aids, developing a realistic agenda, training the recorder, and preparing the meeting room.
- ***Conduct Session:*** Follow agenda to gather and document the project needs and requirements. It is important to ensure all participants are given equal treatment during the process.
- ***Draft the Documents:*** Prepare the formal documents. The information captured in the JAD session is further refined through analysis efforts, open questions or issues discovered through the sessions are resolved, and the final document is returned to stakeholders for review and validation.

Advantages of JAD

- This technique allows for the simultaneous gathering and consolidating of large amounts of information.
- This technique provides a forum to explore multiple points of view regarding a topic.
- This technique produces relatively large amounts of high-quality information in a short period of time.

Disadvantages of JAD

- Requires significant planning and scheduling effort.
- Requires significant stakeholder commitment of time and effort.
- Requires trained and experienced personnel for facilitation and recording.

Some SDLC Methodologies

A number of different SDLC methodologies are used today to guide professionals through their project-based work. In this lecture, I will discuss:

- ✓ Waterfall Model
  - Classical Waterfall Model
  - Parallel Waterfall Model
  - Waterfall V Model
- ✓ Spiral Model
- ✓ Agile Methodologies
  - Scrum
  - eXtreme Programming (XP)
- ✓ Other Practices in Agile:
  - Rapid Application Development (RAD)
  - Rational Unified Process (RUP)

Heavyweight Methodology Vs Lightweight methodology

Heavyweight Methodology

- Heavyweight methodologies are considered to be the traditional way of developing software. These approaches have a tendency to first plan out a large part of the software process in great detail for a long span of time. This approach follows an engineering discipline where the development is predictive and repeatable.
- These methodologies are based on a sequential series of steps, such as requirements definition, solution building, testing and deployment. Heavyweight methodologies require defining and documenting a stable set of requirements at the beginning of a project.
- There are many different heavyweight methodologies but I will limit our discussion to the three most significant methodologies: Waterfall, Spiral Model and Rational Unified Process.

Lightweight methodology

- A lightweight methodology is a software development method that has only a few rules and practices, or only ones that are easy to follow. In contrast, a complex method with many rules is considered a heavyweight methodology.
- There are many different heavyweight methodologies but I will limit our discussion to the three most significant methodologies: Scrum, eXtreme Programming (XP), Rapid Application Development (RAD)

Classical Waterfall Model

- The first formal description of the waterfall model is often cited as a 1970 article by Winston W. Royce, although Royce did not use the term waterfall in that article. The earliest use of the term "waterfall" may have been in a 1976 paper by Bell and Thayer.
- The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. Here, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The development proceeds to the next phase and there is no turning back.
- The model has six phases:

1. Requirement Analysis

- Capture all the requirements.
- Do brainstorming and walkthrough to understand the requirements.
- Do the requirements feasibility test to ensure that the requirements are testable or not.

2. System Design

- As per the requirements, create the design
- Capture the hardware / software requirements.
- Document the designs

3. Implementation

- As per the design create the programs / code
- Integrate the codes for the next phase.
- Unit testing of the code

4. System Testing

- Integrate the unit tested code and test it to make sure if it works as expected.
- Perform all the testing activities (Functional and non-functional) to make sure that the system meets the requirements.
- In case of any anomaly, report it.
- Report your testing activities.

5. System Deployment

- Make sure that the environment is up
- Make sure that the test exit criteria are met.
- Deploy the application in the respective environment.
- Perform a sanity check in the environment after the application is deployed to ensure the application does not break.

6. System maintenance

- Make sure that the application is up and running in the respective environment.
- Incase user encounters and defect, make sure to note and fix the issues faced.
- In case any issue is fixed; the updated code is deployed in the environment.
- The application is always enhanced to incorporate more features, update the environment with the latest feature

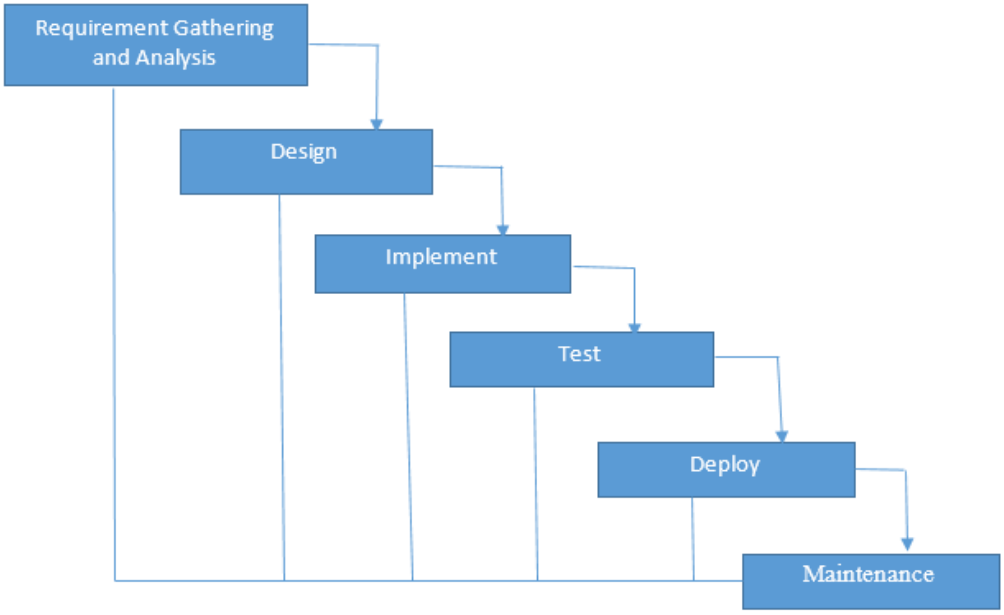


Figure: Classical Waterfall model

Advantages of the 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.

Drawbacks of the Waterfall Model

- It is not desirable for complex project where requirement changes frequently
- Testing period comes quite late in the developmental process
- Documentation occupies a lot of time of developers and testers
- Clients valuable feedback cannot be included with ongoing development phase
- Small changes or errors that arise in the completed software may cause a lot of problems

The waterfall method is still preferred by some developers because of its predictability and known costs. You know what you're paying for and you know when you're going to get it.

When to Use the Waterfall Model

The Waterfall model is appropriate to use in some scenarios.

- When quality is more important than cost or schedule.
- When requirements are very well known, clear, and fixed.
- New version of existing product is needed.
- Porting an existing product to a new platform

Parallel Waterfall Development

- Parallel development reduces the time required to deliver a system.
- Here, instead of doing the design and implementation in sequence, a general design for the whole system is performed. Then the project is divided into a series of subprojects that can be designed and implemented in parallel. Once all subprojects are complete, there is a final integration of the separate pieces, and the system is delivered.
- It also adds a new problem: If the subprojects are not completely independent, design decisions in one subproject may affect another, and at the project end, integrating the subprojects may be quite challenging.

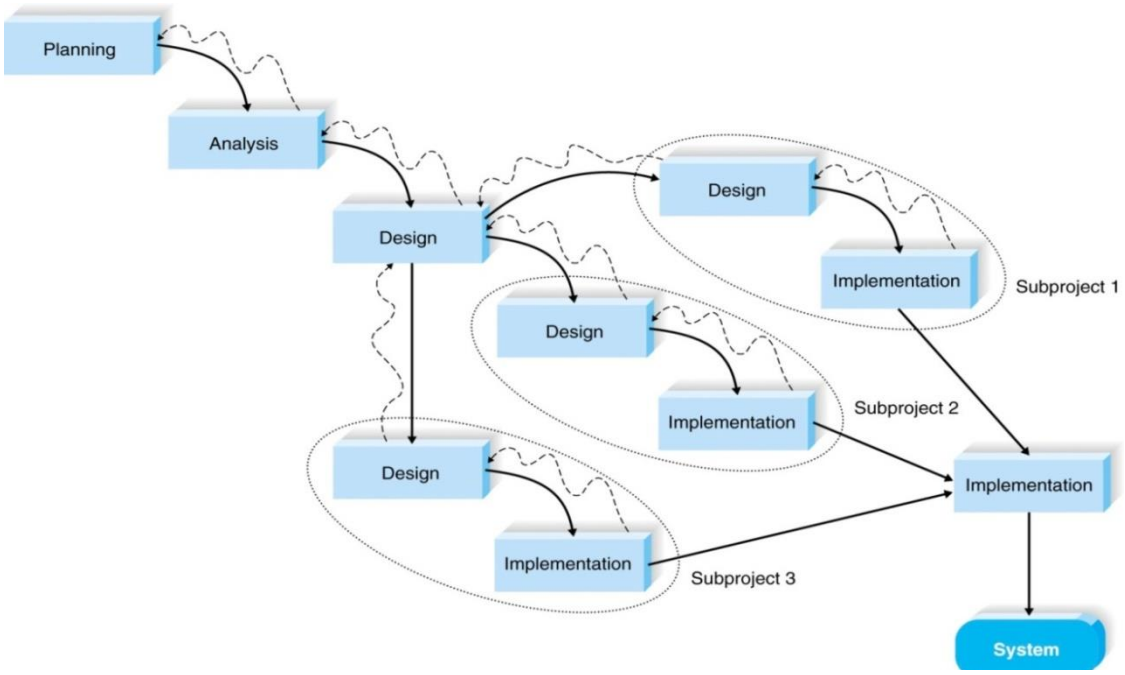


Figure: Parallel Waterfall Development

Waterfall V-Model (Verification and Validation Model)

- V-Model is software development model and V-model is also known as Verification and Validation model.
- **Verification:** It involves static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet.
- **Validation:** It involves dynamic analysis technique (functional, non-functional), testing done by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.

- It is basically extension of waterfall model but unlike water fall model here in V-model, testing is done simultaneously with the development phase. The main disadvantage of water fall model i.e. the Testing Starts after Development ends. But in V-model the testing start early in life cycle.
- The unique of the V-Model is that during each design stage, the corresponding tests are also designed to be implemented later during the testing stages. Thus, during the requirements phase, acceptance tests are designed.

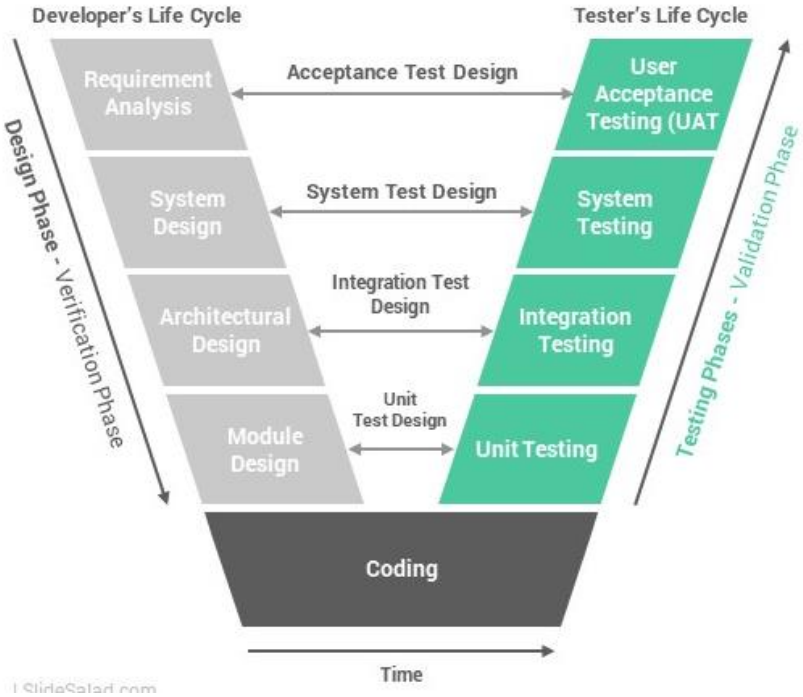


Figure: Waterfall V- Model

There are the various phases of Verification Phase of V-model:

1. ***Business requirement analysis:*** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.

   During the requirements phase, acceptance tests are designed.

2. ***System Design:*** Utilizing feedback and user requirement documents created during the requirements phase, this next stage is used to generate a specification document that will outline all technical components such as the data layers, business logic, and so on.

   System Tests are also designed during this stage for later use.

3. ***Architecture Design:*** During this stage, specifications are drawn up that detail how the application will link up all its various components, either internally or via outside

integrations. Often this is referred to as high-level design. A high-level design provides an overview of a system, product, service or process.

Integration tests are also developed during this time.

4. ***Module Design:*** This phase consists of all the low-level design for the system, including detailed specifications for how all functional, coded business logic will be implemented, such as models, components, interfaces, and so forth. Low-level design is created based on the high-level design such as data structure for a database storage.

   Unit tests should also be created during the module design phase.

5. ***Coding Phase:*** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. ***Unit Testing:*** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. ***Integration Testing:*** Testing devised during the architecture design phase are executed here, ensuring that the system functions across all components and third-party integrations.
3. ***System Testing:*** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.
4. **A*cceptance Testing:*** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

Advantages of the Waterfall V Model

- Less bugs: Do testing in every layer
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

Drawbacks of the Waterfall V-model:

- If any changes happen in midway, then the test documents along with requirement documents has to be updated.
- High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.

When to use the V-model:

- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.

Spiral Model

- ✓ Spiral models initially were suggested in the 1990s by Barry Boehm - a software engineering professor.
- ✓ Spiral Model is a combination of Iterative Development Model and Waterfall Model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.
- ✓ In a few words, Spiral Model can be characterized by repeatedly iterating a set of elemental development processes and eliminating risk, so it is actively being reduced.
- ✓ This model is best used for large projects which involves continuous enhancements. There are specific activities which are done in one iteration (spiral) where the output is a small prototype of the large software. The same activities are then repeated for all the spirals till the entire software is build.
- ✓ To understand how you can get your goals using Spiral Model, let's take a look at this diagram:

Phases of Spiral Model

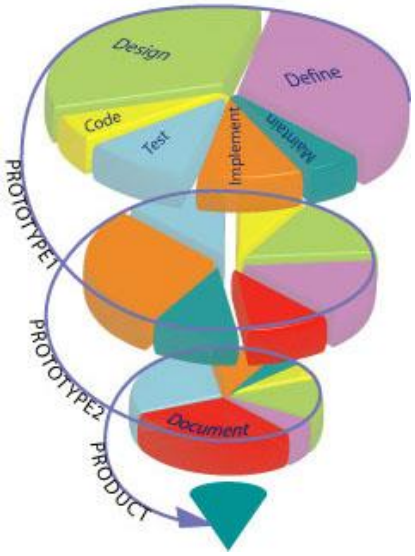
Phase 1: Planning objectives or identify alternative solutions

In this stage, requirements are collected from customers and then the aims are recognized, elaborated as well as analyzed at the beginning of developing the project. If the iterative round is more than one, then an alternative solution is proposed in the same quadrant. It is necessary to define alternatives for implementation (e.g. design A vs. design B) and to determine the framework conditions as well as **costs** or **time expenditure**.

Phase 2: Identifying and resolving the risks

As the process goes to the second quadrant, all likely solutions are sketched, and then the best solution among them gets select. Then the different types of risks linked with the chosen solution

are recognized and resolved through the best possible approach. As the spiral goes to the end of this quadrant, a project prototype is put up for the most excellent and likely solution.



Phase 3: Develop the next level of product

As the development progress goes to the third quadrant, the well-known and mostly required features are developed as well as verified with the testing methodologies. As this stage proceeds to the end of this third quadrant, new software or the next version of existing software is ready to deliver.

Phase 4: Planning the next cycle

As the development process proceeds in the fourth quadrant, the customers appraise the developed version of the project and reports if any further changes are required. At last, planning for the subsequent phase is initiated.

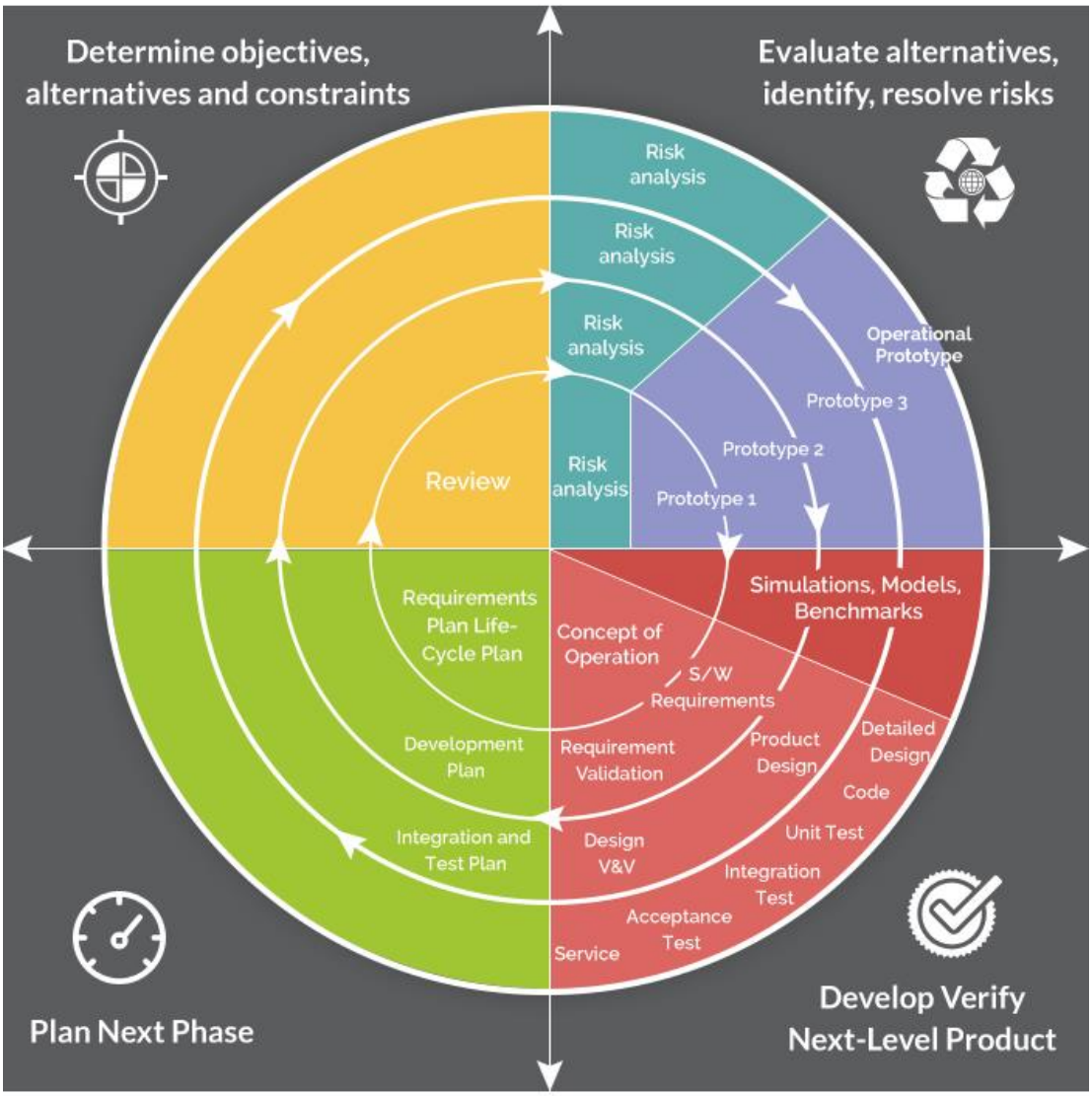


Figure: Spiral models

Advantages of the Spiral Model

- *Suitable for large projects:* Spiral models are recommended when the project is large, bulky or complex to develop.
- *Risk Handling:* There are a lot of projects that have un-estimated risks involved with them. For such projects, the spiral model is the best SDLC model to pursue because it can analyze risk as well as handling risks at each phase of development.
- *Customer Satisfaction:* Customers can witness the development of product at every stage and thus, they can let themselves habituated with the system and throw feedbacks accordingly before the final product is made.
- *Requirements flexibility:* All the specific requirements needed at later stages can be included precisely if the development is done using this model.
- Strong documentation control

Disadvantage of the Spiral Model

- Spiral model is not good for small projects.
- This model is more complex and difficult to understand if a new employee is entered in the project development.
- It can be much expensive.
- Fast development and software is built at the SDLC .
- Not defined end points of the project , so it can take a long time to develop or iterations can be go infinitely.

When to use Spiral Model

- For medium and big projects.
- For high-risk projects.
- Requirements are complex
- Users are unsure of their needs
- If frequent changes required in the project.

Iterative development Approach

- Iterative development is when an attempt is made to develop a product with basic features, which then goes through a refinement process successively to add to the richness in features.
- With each iteration, additional features can be designed, developed and tested until there is a fully functional software application ready to be deployed to customers.
- First, let’s look at simple definitions of the two terms:

Iterative **-** performing repeatedly, i.e. adding new functionality in a repetitive or cyclic manner

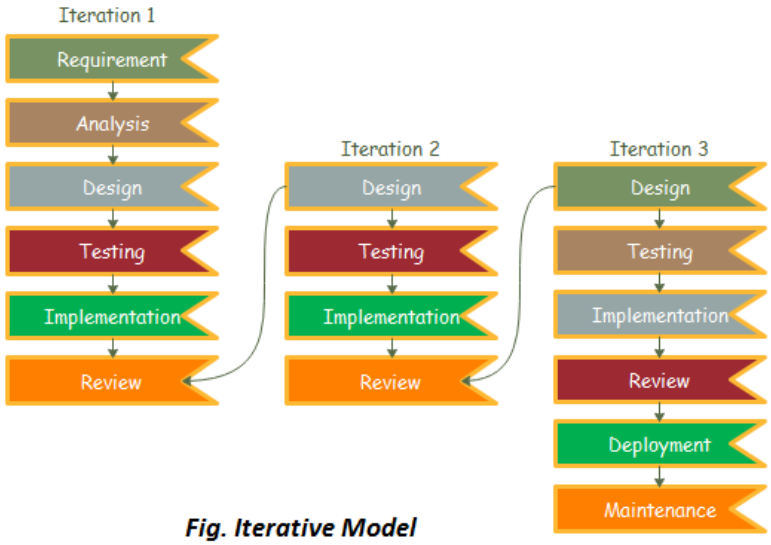


Fig. Iterative Model

An iterative example:

1. In the first iteration of a website check-out engine, payment with debit cards (a first increment) is implemented.
2. The second iteration might produce an increment that supports payment by credit card.
3. Finally, the third iteration might add an increment allowing payment via PayPal.

Incremental development Approach

- In an incremental approach, one aims to build pieces of program/product that is complete in features and richness. In this case, each functionality is built to its fullest and additional functionalities are added in an incremental fashion.
- When we work incrementally we are adding module/portion by module but expect that each module is fully finished.
- First, let's look at simple definitions of the two terms:

Incremental **-** adding new functionality in small chunks

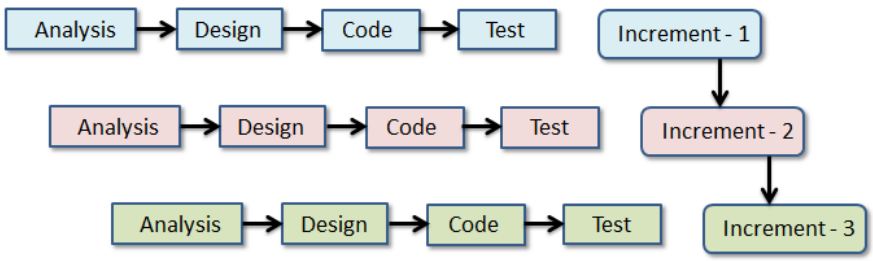


Figure: Incremental Model

Example of Incremental Development:

- I write part one
- I write part two
- I write part three, etc, until the book is finished

Incremental Iteration Approach

Iterative Incremental Development Model is a process where creating requirements, designing, building and testing a system in small parts. This model divided into small parts which will help to make modules very easily. In this model, module passes through some phases like requirements, coding, designing, testing. Iterative Incremental Development model is well known as well as easy to use. Using this model, some part of the system which is built can show to the customer and get some suggestion from it.

This model is useful if we have complete requirements as well as resources to make it. Mostly use for the small project and if customers have requirements to give some output after every phase, so this is a suitable model for them.

An agile methodology mostly depends on building a software product using incremental iterations.

The beauty of the incremental iterative approach is that the main build of the product is always a working product and it can be marketed while the project team is busy creating additional software product features.

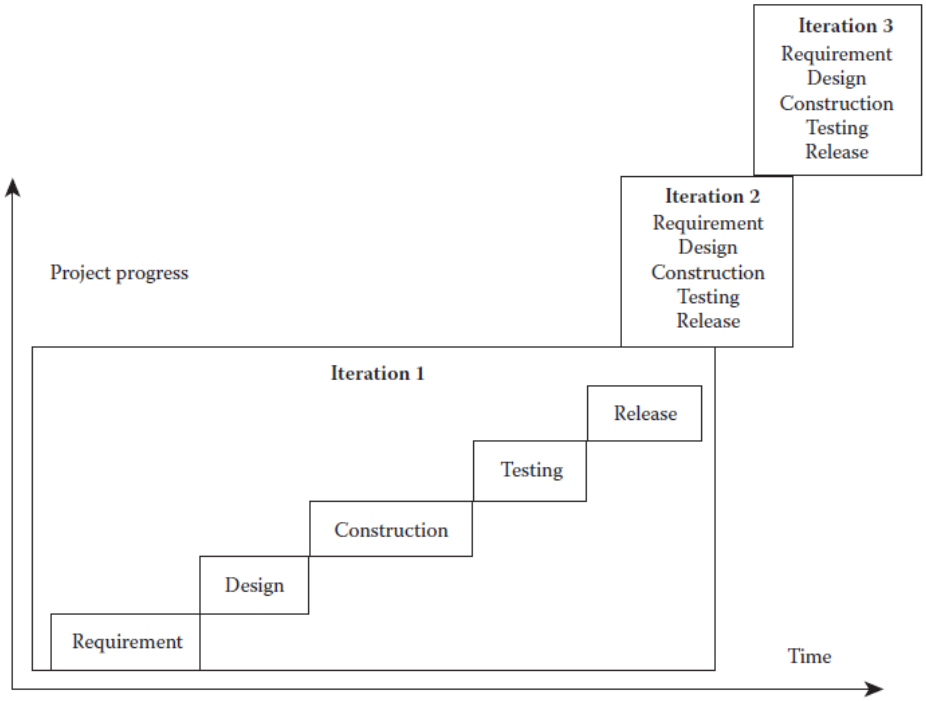


Figure: Incrementally building a software product using iterations

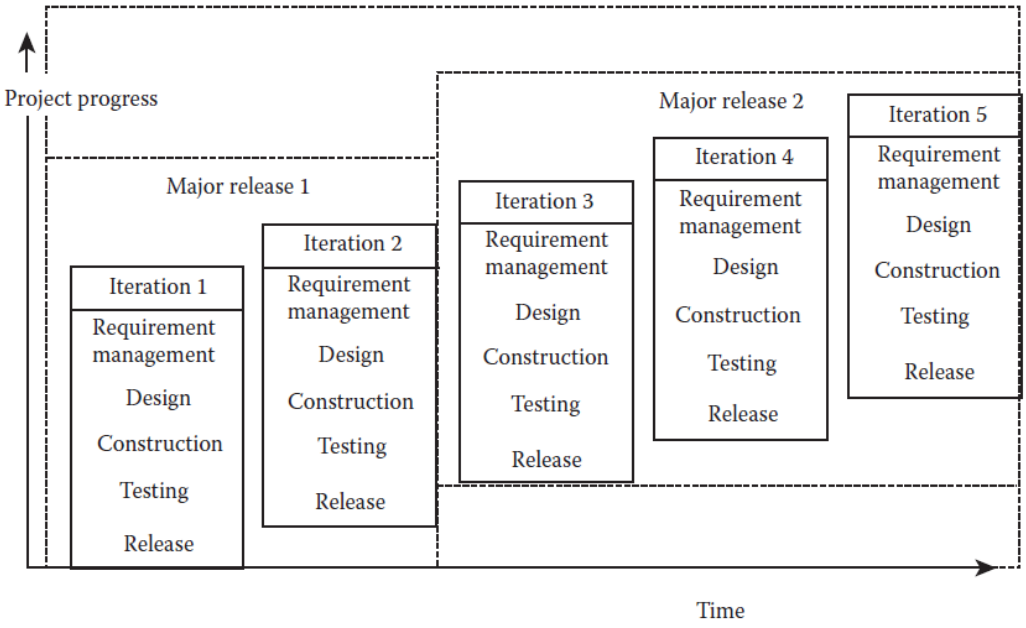


Figure: Complete product development plan in the incremental iteration model

Comparison of Various SDLC Models (Heavyweight Methodology)

Properties of Model	Water Fall Model	Incremental Model	Spiral Model
Planning in early stage	Yes	Yes	Yes
Handle Large-Project	Not Appropriate	Not Appropriate	Appropriate
Detailed Documentation	Necessary	Yes but not much	Yes
Cost	Low	Low	Expensive
Requirement Specifications	Beginning	Beginning	Beginning
Flexibility to change	Difficult	Easy	Easy
User Involvement	Only at beginning	Intermediate	High
Maintenance	Least	Promotes Maintainability	Typical
Risk Involvement	High	Low	Medium to high risk
Framework Type	Linear	Linear + Iterative	Linear + Iterative
Testing	After completion of coding phase	After every iteration	At the end of the engineering phase
Overlapping Phases	No	Yes (As parallel development is there)	No

Properties of Model	Water Fall Model	Incremental Model	Spiral Model
Re-usability	Least possible	To some extent	To some extent
Working software availability	At the end of the life-cycle	At the end of every iteration	At the end of every iteration
Objective	High Assurance	Rapid Development	High Assurance
Team size	Large Team	Not Large Team	Large Team
Customer control over administrator	Very Low	Yes	Yes

Evolution of Agile Approach

- Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.
- Iterative and incremental development methods can be traced back as early as 1957, with evolutionary project management and adaptive software development emerging in the early 1970s. It is a software development approach based on iterative development.
- During the 1990s, a number of lightweight software development methods evolved in reaction to the prevailing heavyweight methods (such as waterfall) that critics described as overly regulated, planned, and micro-managed.
- In 2001, these seventeen software developers met at a resort in Snowbird, Utah to discuss these lightweight development methods. Together they published the Manifesto for Agile Software Development.
- In 2009, a group working with Martin wrote an extension of software development principles, the Software Craftsmanship Manifesto, to guide agile software development according to professional conduct and mastery.
- In 2011, the Agile Alliance created the Guide to Agile Practices (renamed the Agile Glossary in 2016), an evolving open-source compendium of the working definitions of agile practices, terms, and elements, along with interpretations and experience guidelines from the worldwide community of agile practitioners.

Agile Manifesto

The Agile Manifesto is a declaration of the values and principles expressed in agile methodology. Made up for four foundational values and 12 key principles, it aims to help uncover better ways of developing software by providing a clear and measurable structure that promotes iterative development, team collaboration, and change recognition.

The values and principles of the ‘Manifesto for Agile Software Development’ are:

Values

The agile software development emphasizes on four core values.

1 ***Individuals and interactions*** over processes and tools
2 ***Working software*** over comprehensive documentation
3 ***Customer collaboration*** over contract negotiation
4 ***Responding to change*** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Agile Principles

The following 12 Principles are based on the Agile Manifesto.

1. *Satisfy the Customer: Highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

The best ways to ensure you make customers happy while continuously delivering valuable software are to ship early, iterate frequently, and listen to your market continually.

2. *Welcome Change: Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. It's hard not to get a wave of despair when thinking about change requests but change is good if you can react to it fast enough. Change means you are getting closer to client needs and that's a good thing.

3. *Deliver Frequently: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*

Agile philosophy favors breaking a product's development into smaller components and "shipping" those components frequently. Using an agile approach, therefore—and building in more frequent mini-releases of your product—can speed the product's overall development. The sooner you deliver incremental software, the faster the feedback and faster you can identify a wrong turn or a miscommunication with the client.

4. *Work Together: Business people and developers must work together daily throughout the project.*

Business people and developers must work together daily throughout the project. It makes sense for the customer to become part of the team. After all, both the developers and the customers have the same goal; to deliver valuable software. Regular communication between business people and developers helps improve alignment across the organization by building trust and transparency.

5. *Build Projects: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*

A key part of the agile philosophy is empowering individuals and teams through trust and autonomy. The agile team needs to be carefully built to include the right people and skill sets to get the job done, and responsibilities need to be clearly defined before the beginning of a project. Once the work has begun, however, there's no place in agile for micromanagement or hand holding.

6. *Face-To-Face Time: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*

Documenting conversations, creating email narrative streams, even using collaboration software like Slack, are all well and good. But when you're trying to move swiftly, you don't have time to wait for a reply. You need immediate answers, and the only way to achieve that speed of response is by talking to your team member or team in person.

7. *Measure of Progress: Working software is the primary measure of progress.*

That means, is the software working correctly? You're not measuring progress by checking off tasks and moving across your scheduled timeline, but by the success of the software (or whatever) is the subject of your project. The ultimate measure for success is a working product that customers love.

8. *Sustainable Development: Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.*

Keeping up with a demanding, rapid release schedule can be taxing on a team. Especially if expectations are set too high. Agile principles encourage us to be mindful of this and set realistic, clear expectations. The idea is to keep morale high and improve work-life balance to prevent burnout and turnover among members of cross functional teams.

9. *Continuous Attention: Continuous attention to technical excellence and good design enhances agility.*

Whether you're working on code or something more concrete, you want to make sure that after each iteration it's improving. You don't want to have to come back and fix things later. Fix them now. Better still, make sure they're getting better. Use scrum, an agile framework for completing complex projects, to help review and keep the project evolving.

10. *Keep It Simple: Simplicity—the art of maximizing the amount of work not being done—is essential.*

If you're looking to move quickly through a project, then you're going to want to cut out unnecessary complexities. Keeping things as simple as possible is a great ethic to streamline your process. You can do this many ways, including the use of agile tools (Gantt Chart for big picture, Dashboards for Reporting on Progress, etc.) that cut out the busy work and give you more control over every aspect of the project.

11. *Organized Teams: The best architectures, requirements and designs emerge from self-organizing teams.*

Agile principles suggest the use of self-organizing teams which work with a more "flat" management style where decisions are made as a group rather than by a singular manager or management team. When you have a strong team, you want to give that team the autonomy to act independently. This means they can adapt to change quicker. In fact, they can do everything with greater agility because you've given them the trust to act without second guessing them. If you've done your job in collecting the right people, then they'll do their job addressing issues and resolving them before they become problems.

12. *Reflect for Effectiveness: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

We've all been on projects that end with an AAR, After Action Review. These reviews help the next project but not the one you just finished. Agile defines several ceremonies and important among those is the Retrospective. Generally held at the end of each Sprint/Iteration it is a way for teams to catch and improve behaviors before they have a huge, detrimental impact on the project.

Stages of the Agile Life Cycle

The main steps for Agile project management are as follows:

- In the Agile methodology, each project is broken up into several 'Iterations'.
- All Iterations should be of the same time duration (between 2 to 8 weeks).
- At the end of each iteration, a working product should be delivered.
- In simple terms, in the Agile approach the project will be broken up into 10 releases (assuming each iteration is set to last 4 weeks).
- Rather than spending 1.5 months on requirements gathering, in Agile software development, the team will decide the basic core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be taken up in the next iteration or subsequent iterations, based on priority.
- At the end of the first iterations, the team will deliver a working software with the features that were finalized for that iteration.
- There will be 10 iterations and at the end of each iteration the customer is delivered a working software that is incrementally enhanced and updated with the features that were shortlisted for that iteration.

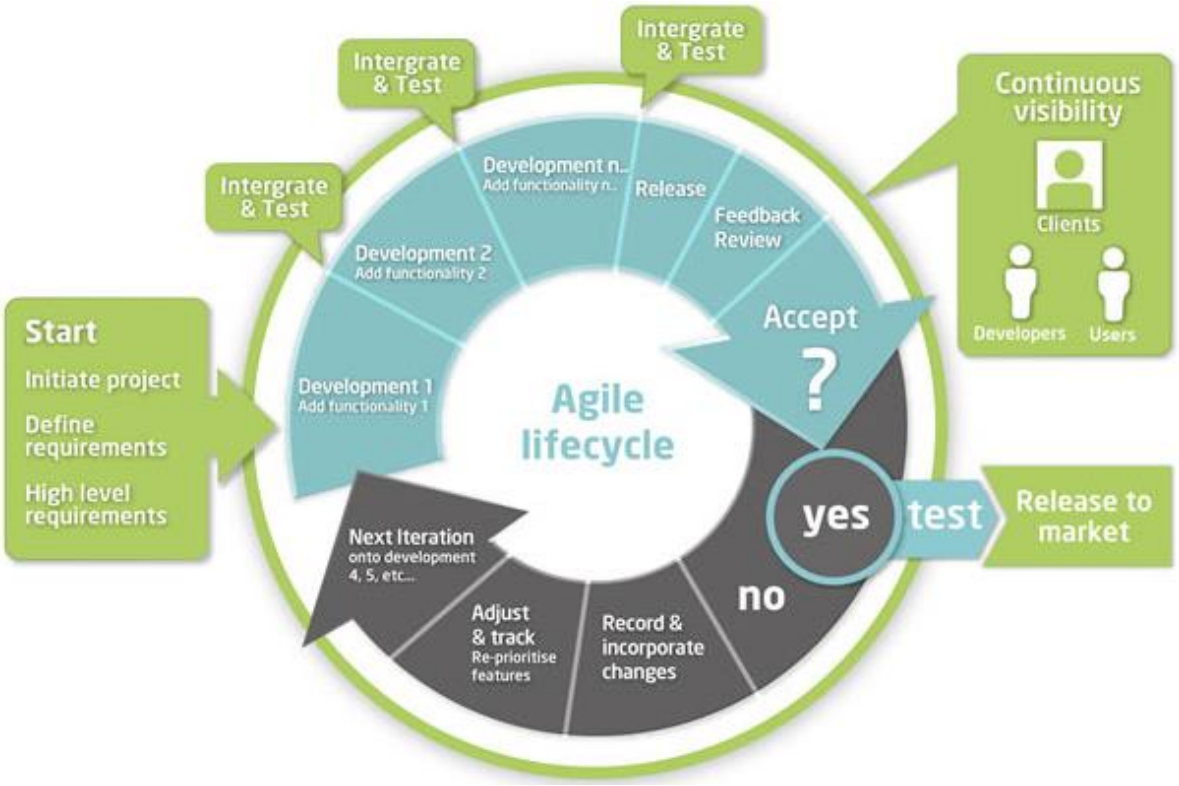


Figure: Agile model

Advantages of the Agile Model

- Continuous attention to technical excellence and good design.
- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Regular adaptation to changing circumstances. Even late changes in requirements are welcomed

Drawbacks of the Agile Model

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process.

Popular Agile Methodologies/Frameworks

There are many popular agile frameworks used by various organizations. Often these organizations modify parts of the frameworks as they see fit and as they iterate on their own agile processes. Below you'll find our overviews of several commonly used and well-document frameworks for agile software development.

Methodologies That Are Used to Implement Agile:
Agile is a framework and there are a number of specific methods within the Agile movement. You can think of these as different flavors of Agile:

- Scrum [1995]
- eXtreme Programming (XP) [1996]
- Dynamic Systems Development Method (DDSM) [1994]
- Feature Driven Development (FDD) [1997]
- Adaptive Software Development (ASD)
- The Crystal Method [1996]
- Lean Software Development (LSD)
- Disciplined Agile (DA)
- Scaled Agile Framework (SAFe)

Other Practices in Agile:
There are many other practices and frameworks that are related to Agile. They include:

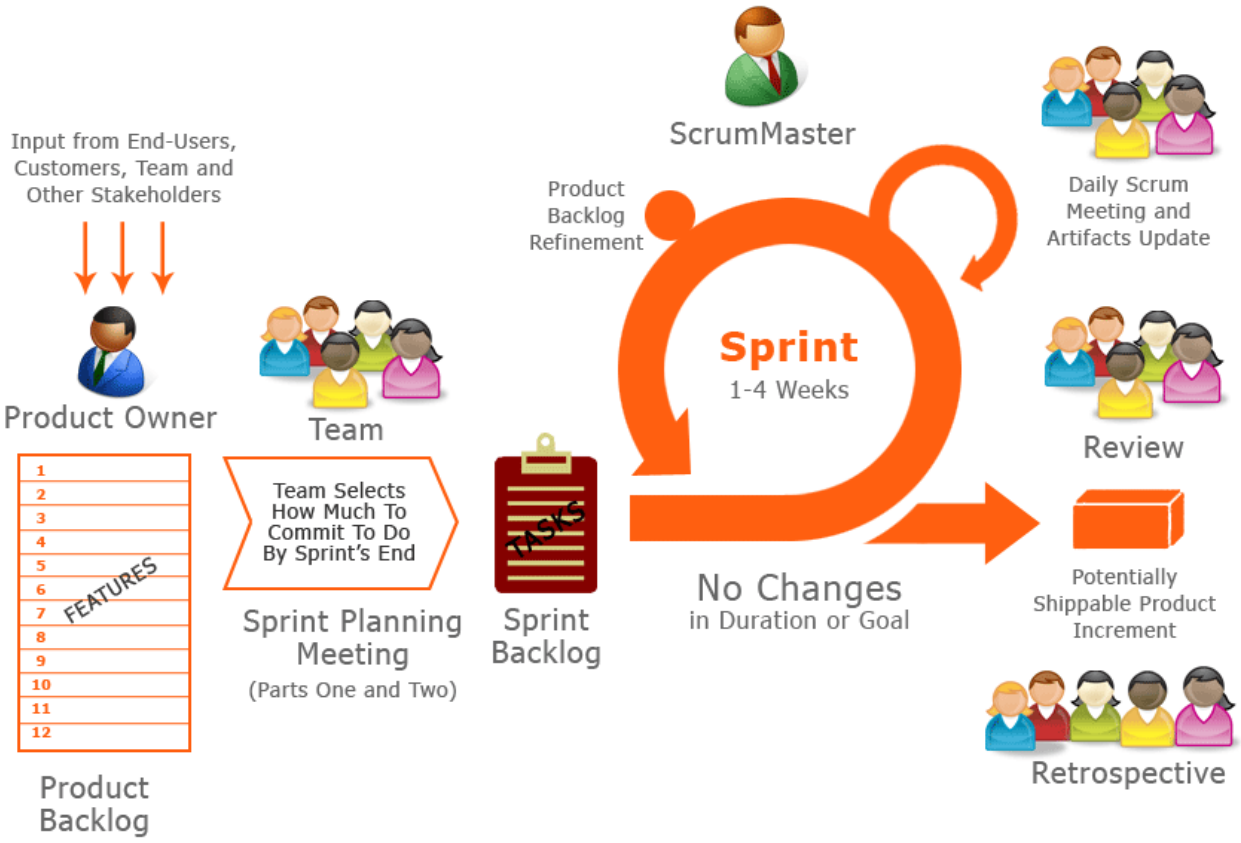
- Rapid Application Development (RAD) [1991]
- Rational Unified Process (RUP) [1994]

Agile Methodology Vs Traditional Methodology

Issue	Traditional Methodology	Agile Methodology
Development cycle	Incremental	Linear
Requirements	Clearly defined	Not defined
Documentation	Detailed / heavy	Light
Team members	Distributed teams	Co-location of teams
Development style	Predictive	Adaptive
Client involvement	Low	Active
Project Size	Large	Small
Domain	Predictable	Unpredictable
Team size	Large	Small
Return on investment	End of project	Early in the project

Scrum

The goal of Scrum is to dramatically improve productivity in teams and to deliver new software every 2-4 weeks. Agile Scrum methodology involves forming teams with diversity, strong communicative teamwork, frequent feedbacks from clients, a lot of participates of users etc.



Some terminologies related to Scrum are discussed in the following sections.

Scrum Events

Scrum activities are driven through scrum events.

Sprint: The Sprint is a timebox of one month or less during which the team produces a potentially shippable product Increment.

Sprint Planning: Sprint planning meeting is first attended by the customers, users, management, Product owner and Scrum Team where a set of goals and functionality are decided on. Next the Scrum Master and the Scrum Team focus on how the product is implemented during the Sprint.

Daily Scrum: It is a daily meeting for approximately 15 minutes, which are organized to keep track of the progress of the Scrum Team and address any obstacles faced by the team.

Sprint Review: At the end of the Sprint, the entire team (including product owner) reviews the results of the sprint with stakeholders of the product to get feedback.

Sprint Retrospective: At the end of the Sprint following the sprint review the team (including product owner) should reflect upon how things went during the previous sprint and identify adjustments they could make going forward. The result of this retrospective is at least one action item included on the following Sprint's Sprint Backlog.

Artifacts

Product Backlog: A product backlog is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome. The Product Owner is responsible for maintaining the Product Backlog.

Sprint Backlog: Sprint backlog is a subset of product backlog. It contains the requirements that are used to develop the current sprint.

Increment: The increment is the collection of the Product Backlog Items that meet the team's Definition of Done by the end of the Sprint. The Product Owner may decide to release the increment or build upon it in future Sprints.

Scrum Roles

- ***Scrum Master*:** Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- ***Product owner:*** The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- ***Scrum Team:*** Team manages its own work and organizes the work to complete the sprint or cycle

Scrum Lifecycle

The Scrum Lifecycle starts with a prioritized backlog, but does not provide any guidance as to how that backlog is developed or prioritized.

The Scrum Lifecycle consists of a series of Sprints, where the end result is a potentially shippable product increment. Below is a description of the key steps in the Scrum Lifecycle:

1. Establish the Product Backlog.
2. The product owner and development team conduct Sprint Planning. Determine the scope of the Sprint in the first part of Sprint Planning and the plan for delivering that scope in the second half of Sprint Planning.
3. As the Sprint progresses, development team perform the work necessary to deliver the selected product backlog items.
4. On a daily basis, the development team coordinate their work in a Daily Scrum.

5 At the end of the Sprint the development team delivers the Product Backlog Items selected during Sprint Planning. The development team holds a Sprint Review to show the customer the increment and get feedback. The development team and product owner also reflect on how the Sprint has proceeded so far and adapting their processes accordingly during a retrospective.

6 The Team repeats steps 2–5 until the desired outcome of the product have been met.

Advantages of Scrum

- Scrum methodology enables projects where the business requirements documentation is hard to quantify to be successfully developed.
- It is a lightly controlled method which insists on frequent updating of the progress in work through regular meetings. Thus there is clear visibility of the project development.
- Like any other agile methodology, this is also iterative in nature. It requires continuous feedback from the user.
- Due to short sprints and constant feedback, it becomes easier to cope with the changes.
- Daily meetings make it possible to measure individual productivity. This leads to the improvement in the productivity of each of the team members.
- Issues are identified well in advance through the daily meetings and hence can be resolved in speedily.
- It is easier to deliver a quality product in a scheduled time.
- The overhead cost in terms of process and management is minimal thus leading to a quicker, cheaper result.

Disadvantages of Scrum

- If the team members are not committed, the project will either never complete or fail.
- It is good for small, fast moving projects as it works well only with small team.
- This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed in time.
- If any of the team members leave during a development it can have a huge inverse effect on the project development
- Project quality manager is hard to implement and quantify unless the test team are able to conduct regression testing after each sprint.

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. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes on team building many times each day. The term 'extreme' comes from taking these commonsense principles and practices to extreme levels. A summary of XP terms and practices is shown below:

- ***Planning*** – In the planning stage user stories are written. User stories are used to create time estimates for release planning meeting. The programmer estimates the effort needed for implementation of customer stories and the customer decides the scope and timing of releases based on estimates.
- ***Small/short releases*** – An application is developed in a series of small, frequently updated versions. New versions are released anywhere from daily to monthly.
- ***Metaphor*** – Metaphor is a simple shared story of how the system works. The system is defined by a set of metaphors between the customer and the programmers.
- ***Simple Design*** – The emphasis is on designing the simplest possible solution that is implemented and unnecessary complexity and extra code are removed immediately.
- ***Refactoring*** – It involves restructuring the system by removing duplication, improving communication, simplifying and adding flexibility but without changing the functionality of the program
- ***Pair programming*** – All production code are written by two programmers on one computer.
- ***Collective ownership*** – No single person owns or is responsible for individual code segments rather anyone can change any part of the code at any time.
- ***Continuous Integration*** – A new piece of code is integrated with the current system as soon as it is ready. When integrating, the system is built again and all tests must pass for the changes to be accepted.
- ***40-hour week*** – No one can work two overtime weeks in a row. A maximum of 40-hour working week otherwise it is treated as a problem.
- ***On-site customer*** – Customer must be available at all times with the development team.
- ***Coding Standards*** – Coding rules exist and are followed by the programmers so as to bring consistence and improve communication among the development team.

The lifecycle of XP

The lifecycle of an XP is divided into six phases: Exploration, Planning, Iterations to release, Production, Maintenance and Death.

- In the ***Exploration phase***, the customer writes out the story cards they wish to be included in their program.
- This leads to ***Planning phase*** where a priority order is set to each user story and a schedule of the first release is developed.
- Next in the ***Iterations to Release phase***, the development team first iteration is to create a system with the architecture of the whole system then continuously integrating and testing their code.
- Extra testing and checking of the performance of the system before the system can be released to the customer is done in the ***Production phase***.
- Postponed (rest of) ideas and suggestions found at this phase are documented for later implementation in the updated releases made at the ***Maintenance phase***. Also, customer feedback is considered.
- Finally the ***Death Phase*** is near when the customer have no more stories to be implemented and all the necessary documentation of the system is written as no more changes to the architecture, design or code is made.

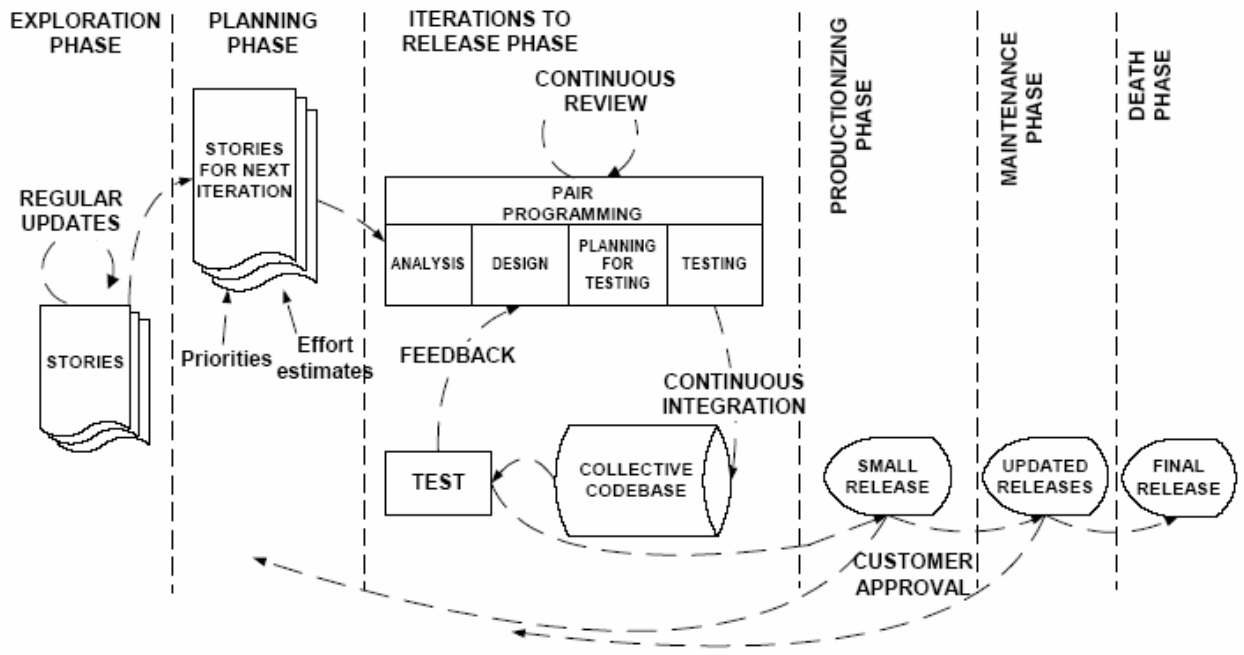


Figure: Life Cycle of eXtreme Programming Method

Advantages of XP

- The greatest advantage of eXtreme Programming is that this methodology allows software development companies to save costs and time required for project realization.
- One of the major advantages of eXtreme Programming is that it reduces the risks related to programming or related to project failure.
- Simplicity is one more advantage of eXtreme Programming projects. The developers who prefer to use this methodology create extremely simple code that can be improved at any moment.
- Constant feedback; demonstrate the software early and often, listen carefully and make any changes needed..
- eXtreme Programming helps increase employee satisfaction and retention. eXtreme Programming is a value-driven approach that sets fixed work time, with little scope for overtime.
- Teamwork; everyone is part of the team. Team members work together on everything from requirements to code. Developers work in pairs; pair programming and never feel alone or forgotten.

Drawbacks of XP

- Lack of defect documentation may lead to the occurrence of similar bugs in the future.
- XP is not the best option if programmers are separated geographically.
- Not recommended for team with more than ten members
- Required experienced developers
- Lack of documentation

Comparison of XP and Scrum

Features	**eXtreme Programming**	**Scrum**
Development Approach	Iterative and incremental	Iterative and incremental
Project Size	Small	All
Team Size	2 to 10	Multiple teams of less than 10 members
Iteration/Sprint Duration	1 to 3 weeks	4 weeks
Project Management	No	Yes; Practices for project management are available
Response to Change	Quick	Quick
Documentation	Less	Less
Design Flexibility	Start from Simple design that can be changed using refactoring	Focus on simple design
Changes During Iteration	Allowed	Not allowed
Feedback	Span from minutes to months	Span over a month
Testing	Unit testing, integration testing, acceptance testing	Not defined
Coding Standards	Properly defined	Not defined

Other Practices in Agile:

Rapid Application Development (RAD)

- ✓ Rapid Application Development is heavily focused primarily on rapid prototyping of software products, frequently iterating based on feedback, and continuously releasing updated versions of those products to the market.
- ✓ The RAD concept was officially introduced to the public in 1991 with the book Rapid Application Development by James Martin. Rapid application development has become one of the most popular and powerful development methods, which falls under the parental category of agile development techniques.

The James Martin approach to RAD divides the process into four distinct phases:

1. ***Requirements planning phase*** – Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.
2. ***User design phase*** – The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. *User Design* is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.
3. ***Construction phase*** – focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.
4. ***Cutover phase*** – resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.

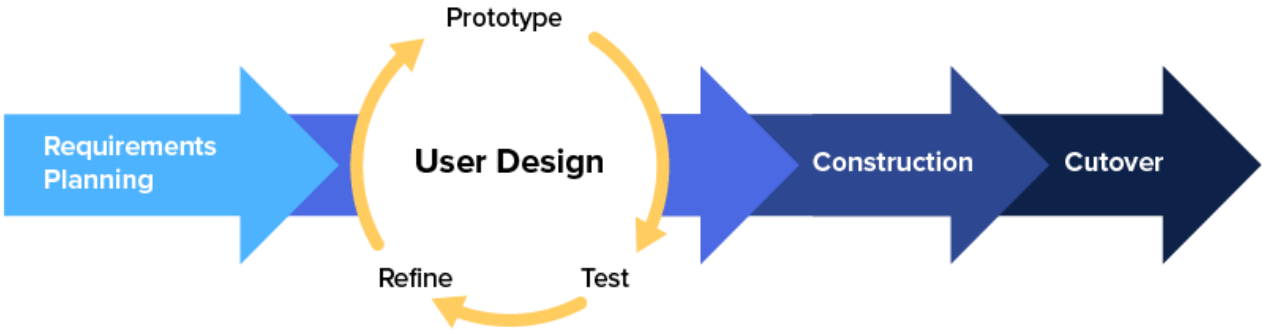


Figure: Phases in the James Martin approach to RAD

Advantages of RAD Model

- The time required to develop the software is drastically reduced due to a reduced requirement analysis business requirements documentation and software requirement specification) and planning stage.
- All the software prototypes produced can be kept in a repository for future use. If a component is being picked for the repository, it is already tested and hence need not be tested again. This helps in saving time required for testing.
- It is a big cost saver in terms of project budget as well as project time and cost due to reusability of the prototypes.
- It is much easier for a project manager to be accurate in estimating project costs which of course means that project cost controls are easier to implement and manage as well.
- Customer giving feedback in the whole process. Hence the end user satisfaction level is higher when the end result is produced.
- It promotes better documentation through written test cases.

Drawbacks of RAD Model

- A technically strong team is essential to adequately identify and deliver business requirements.
- Documentation is completed in the final phase, so problems and progress are harder to track, which significantly impacts scalability.
- The RAD model requires a frequent cycle of prototypes, and consequently, all stakeholders must be willing and able to commit to regular meetings to communicate and provide feedback frequently.
- It is difficult for large-scale projects.

When to use the RAD Model

- RAD is particularly useful for small businesses delivering innovative products in a competitive market place.
- The on-the-fly approach accommodates unexpected changing of requirements.
- RAD models can be very successful when a quick delivery of a product is needed for a customer. It is also the best model to choose when there are going to be changes made to the prototype throughout the process before the final product is completed.
- RAD should only be used when a system can be modulated to be delivered incrementally. If you need to build an internal business tool or a customer facing portal, RAD can assist you to deliver better experience to your end users.

Rational Unified Process (RUP) Model

All efforts, including modeling, is organized into workflows in the Rational Unified Process (RUP) and is performed in an iterative and incremental manner. The lifecycle of the RUP is presented in Figure. Some of the key features of the RUP are as follows :

- It uses a component based architecture which creates a system that is easily extensible, promotes software reuse and intuitively understandable. The component commonly being used to coordinate object oriented programming projects.
- Uses visually modeling software such as UML – which represent its code as a diagrammatic notation to allow less technically competent individuals who may have a better understanding of the problem to have a greater input.
- Manage requirements using use-cases and scenarios have been found to be very effective at both capturing functional requirements and help in keeping sight of the anticipated behaviors of the system.
- Design is iterative and incremental – this helps reduce project risk profile, allows greater customer feedback and help developers stay focused.
- Verifying software quality is very important in a software project. UP assists in planning quality control and assessment built into the entire process involving all member of the team.

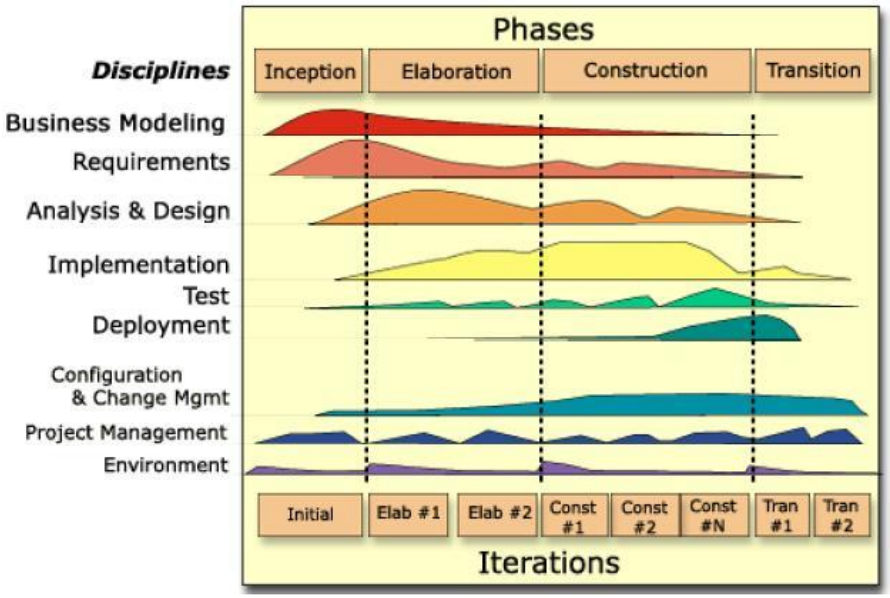
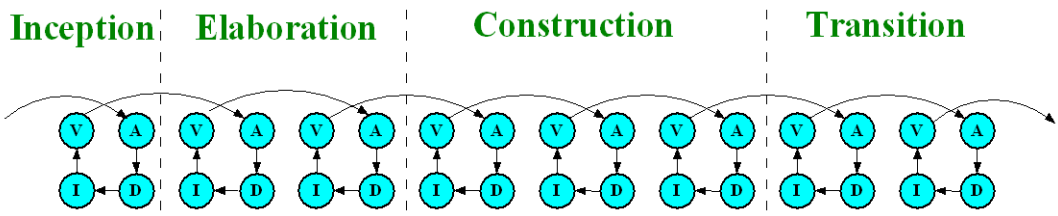


Figure: the overall architecture of the RUP

In the RUP, all progress is made as continual ADIV cycles

- ADIV: Analysis, Design, Implementation, Validation



These key features above are guidelines to be adhered throughout a projects' lifecycle. To determine the length of the project, RUP divides the project into four phases which are shown above in Figure of RUP and discussed below:

- ***Inception*** – By the end of this process a business case should have been made; feasibility of the project assessed; and the scope of the design should be set.
- ***Elaboration*** – In this phase a basic architecture should have been produced and a plan of construction agreed. Furthermore, a risk analysis takes place and those risks considered to be major should have been addressed.
- ***Construction*** – This process produces a beta-release system. A working system should be available and sufficient enough for preliminary testing under realistic conditions.
- ***Transition*** – The system is introduced to the stakeholders and intended users. It is crossed when the project team and the stakeholders agree that the objectives agreed in the inception phase have been met and the user is satisfied.

The black heaps on the graph show the amount of work to be done for a workflow over a period. For example, let us consider the black heap for the business modeling workflow. It shows a large amount of work at the beginning of the project. This means the business modeling work is done early in the project. A major part of business modeling is done in the inception phase. It also shows that work is spread over Iterations 1 and 2 for the elaboration phase.

Advantages of RUP

- When a phase changes, through the use of a workflow, some work can be carried out in that other phase. For example, the requirements workflow can be carried out even during the construction phase of the project.
- Thus, overlapping of work is possible over the phases of the project. This kind of functionality is not available in the traditional Waterfall model.
- It is proactively able to resolve the project risks associated with the client's evolving requirements requiring careful change request management
- Less time is required for integration as the process of integration goes on throughout the software development life cycle.

Drawbacks of RUP

- RUP brings an overhead in terms of maintaining the project phases as well as workflows. This kind of functionality is not needed on projects where the requirements are not expected to change. For these projects, usage of RUP is overkill.
- 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.
- Integration throughout the process of software development, in theory sounds a good thing. But on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing

When to Use RUP

- This model is best suited for projects where the requirements from the end users are ambiguous at the beginning because the end users themselves may not be clear as to what they expect in the software product. RUP allows the refinement of the requirements over several iterations.
- RUP is definitely suited for software projects in which changes occur in the design or requirements. RUP takes care of these changes by incorporating them during iterations.

How to Choose the Best SDLC Model: Traditional SDLC vs. Agile

With so many different approaches to structuring software processes within your organization, you're probably wondering how to go about choosing one. There are many factors that may influence which framework you choose to work with. Such as:

- Company size
- Team structure and their skill
- Available resources
- Needs of stakeholders
- Structure/size of your product portfolio

Each Methodology has its own unique set of strengths and weaknesses. And the framework that works for someone else's team might not be the right one for you. Ultimately, you'll have to experiment a bit and figure out what works best for you.

It is no doubt that the adoption of the right technology in any organization has substantial benefits. However, it is not every software or application you find on the marketplace will solve your needs. It is advisable to align your software adoption to the specific needs and conditions at your organization. This is one of the benefits of the Agile software development lifecycle, as this method aims at enhancing productivity and satisfying customers' needs. It is the best consideration for startups, as it has a room for this flexibility. In addition, startups should consider Agile SDLC because it provides an avenue for them to use their limited resources in a smart way to gain the competitive advantage they need. However, for Agile SDLC to work effectively, you need the input of the right dedicated development team.