Adaptive Software Development

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

Adaptive Software Development is a move towards adaptive practices, leaving the deterministic practices in the context of complex systems and complex environments. Adaptive Software Development focuses on collaboration and learning as a technique to build complex systems. It is evolved from the best practices of Rapid Application Development (RAD) and Evolutionary Life Cycles.

Audience

Adaptive Software Development is written for project teams that have been struggling with high-speed, high-change projects and are looking for ways to improve performance and to moderate burnout, especially as the projects they undertake get larger and the teams become more distributed.

Prerequisites

Before you start proceeding with this tutorial, we are assuming that you are already aware about the basics of Software Development Life Cycle. If you are not well aware of these concepts, then we will suggest you to go through our short tutorials on SDLC.

Copyright & Disclaimer

Table of Contents

1. ASD – Introduction

What is Agile?

In literary terms, the word "agile" means someone who can move quickly and easily or someone who can think and act quickly and clearly. In business, "agile" is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job. Business "agility" means that a company is always in a position to take account of the market changes.

In software development, the term "agile" is adapted to mean "the ability to respond to changes – changes from Requirements, Technology and People."

Agile Manifesto

The Agile Manifesto was published by a team of software developers in 2001, highlighting the importance of the development team, accommodating changing requirements and customer involvement.

The Agile Manifesto is:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value-

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- 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.

Characteristics of Agility

Following are the characteristics of Agility-

- Agility in Agile Software Development focuses on the culture of the whole team with multi-discipline, cross-functional teams that are empowered and self-organizing.
- It fosters shared responsibility and accountability.
- Facilitates effective communication and continuous collaboration.
- The whole-team approach avoids delays and wait times.
- Frequent and continuous deliveries ensure quick feedback that in in turn enable the team align to the requirements.

- Collaboration facilitates combining different perspectives timely in implementation, defect fixes and accommodating changes.
- Progress is constant, sustainable, and predictable emphasizing transparency.

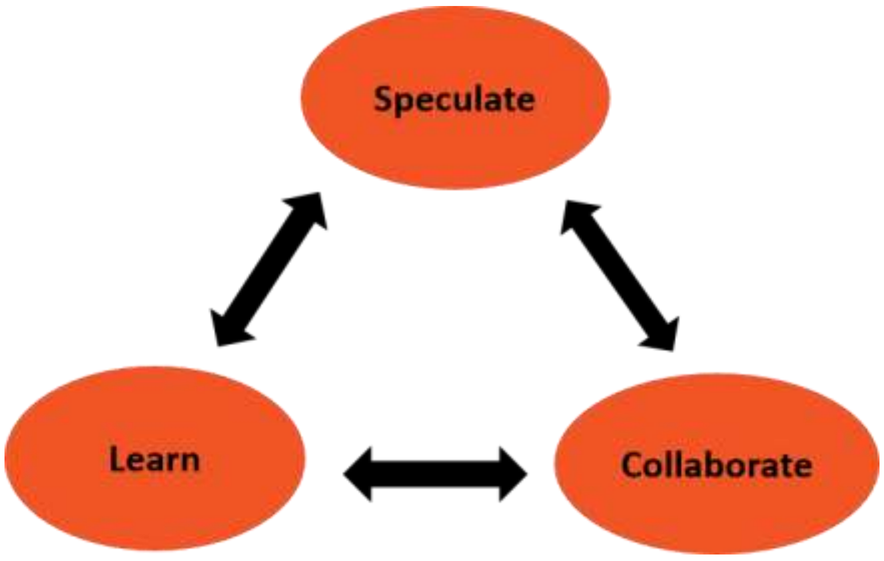
Agile Methodologies

Early implementations of Agile methods include Rational Unified Process, Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM). These are now collectively referred to as the Agile methodologies, after the Agile manifesto was published in 2001.

In this tutorial, we will learn the Agile Methodology – **Adaptive Software Development.**

What is Adaptive Software Development?

Adaptive Software Development is a move towards adaptive practices, leaving the deterministic practices in the context of complex systems and complex environments. Adaptive Software Development focuses on collaboration and learning as a technique to build complex systems. It is evolved from the best practices of Rapid Application Development (RAD) and Evolutionary Life Cycles. Adaptive Software Development was then extended to include adaptive approaches for the management, with speculation replacing Planning.



Jim Highsmith published a book on Adaptive Software Development in 2000. In Highsmith's words –

> "Adaptive Software Development is cyclical like the evolutionary model, with the phase names Speculate, collaborate, learn reflecting the unpredictable realm of increasingly complex systems. Adaptive development goes further than its evolutionary heritage in two key ways. First, it explicitly replaces determinism with emergence. Second, it goes beyond a change in Life Cycle to a deeper change in management style."

2. SDLC Models – Evolution

A Software Development Life Cycle (SDLC) model is a framework that describes the activities performed at each stage of a software development project.

In a Software Development Life Cycle, the activities are performed in five phases-

- **Requirements Gathering:** Requirements for a software to be developed are gathered. These requirements will be in a language that is understood by the customer / user. Domain specific terminology is recommended.

- **Analysis:** The gathered requirements are analyzed from implementation point of view and the software specifications are written to cover both, the functional requirements and the non-functional requirements.

- **Design:** This phase involves arriving at the software architecture and implementation specifics based on technology chosen for development.

- **Construction:** In this phase, the code is developed, unit tested, integrated, integration tested and the build is produced.

- **Testing:** Functional testing of the built software is done in this phase. This also includes the testing of non-functional requirements.

There are two approaches to performing these activities-

- **Prescriptive**: The SDLC models that will provide you ways of performing the activities in a prescribed manner as defined by the framework.

- **Adaptive**: The SDLC models that will give you flexibility in performing the activities, with certain rules that need to be followed. The agile methods mostly follow this approach, with each one having its rules. However, following an adaptive or agile approach does not mean that the software is developed without following any discipline. This would lead to a chaos.

You need to understand that we cannot say that a specific SDLC model is good or bad. Each of them has its own strengths and weaknesses and thus are suitable in certain contexts.

When you choose an SDLC model for your project, you need to understand-

- Your Organization Context
- Your Technology Context
- Your Team Composition
- Your Customer Context

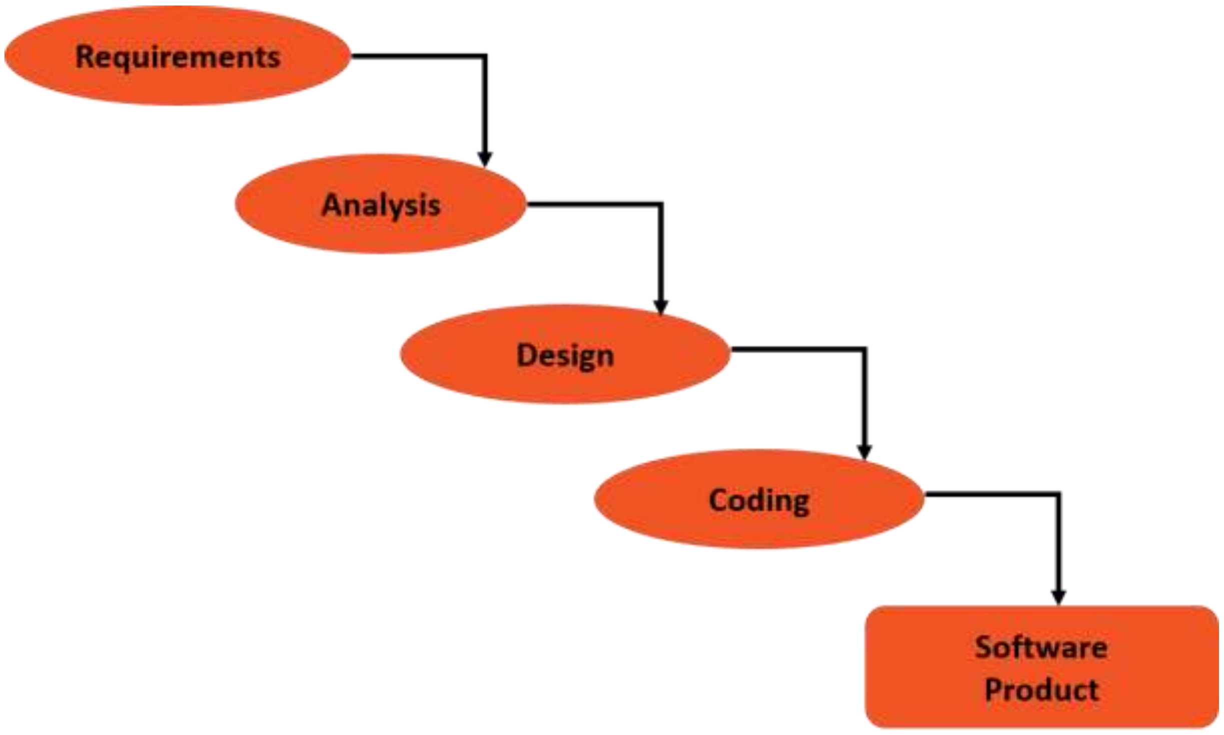
For example, if the software development is predictable, you can use a Prescriptive approach. On the other hand, if the software development is unpredictable, i.e. requirements are not entirely known, or the development team does not have prior exposure to the current domain or technology, etc. then Adaptive approach is the best choice.

In the following sections, you will understand the most prevalent SDLC models that are evolved during the execution of software development projects across the industry. You will also get to know the strengths and weaknesses of each of them and in what contexts they are suitable.

3. SDLC – Waterfall Model

The Waterfall model is a classic SDLC model that is widely known, understood and commonly used. It was introduced by Royce in 1970 and is still being followed as a common approach for software development in various organizations across the industry.

In Waterfall model, each lifecycle phase can start only after the earlier lifecycle phase is complete. Thus, it is a linear model with no feedback loops.



Waterfall Model – Strengths

The strengths of the Waterfall model are-

- Easy to understand, easy to use.
- Provides structure to inexperienced development team.
- Milestones are well understood.
- Sets requirements stability.
- Ideal for management control (planning, monitoring, reporting).
- Works well when quality is more important than cost or schedule.

Waterfall Model – Weaknesses

The weaknesses or the disadvantages of the Waterfall model are-

- Idealised – It does not match reality well.
- Unrealistic – cannot expect accurate requirements early in the project.
- Does not reflect iterative nature of exploratory development that is more common.
- Difficult and expensive to make changes.
- Software is delivered only at the end of the project. Due to this -
    - Delays discovery of serious defects.
    - Possibility of delivery of obsolete requirements.
- Significant management overhead, which can be costly for small teams and projects.
- Requires experienced resources at every phase – analysts, designers, developers, testers.
- Testing starts only after the development is complete and the testers are not involved in any of the earlier phases.
- The expertize of the cross-functional teams is not shared as each phase is executed in silos.

When to Use Waterfall Model?

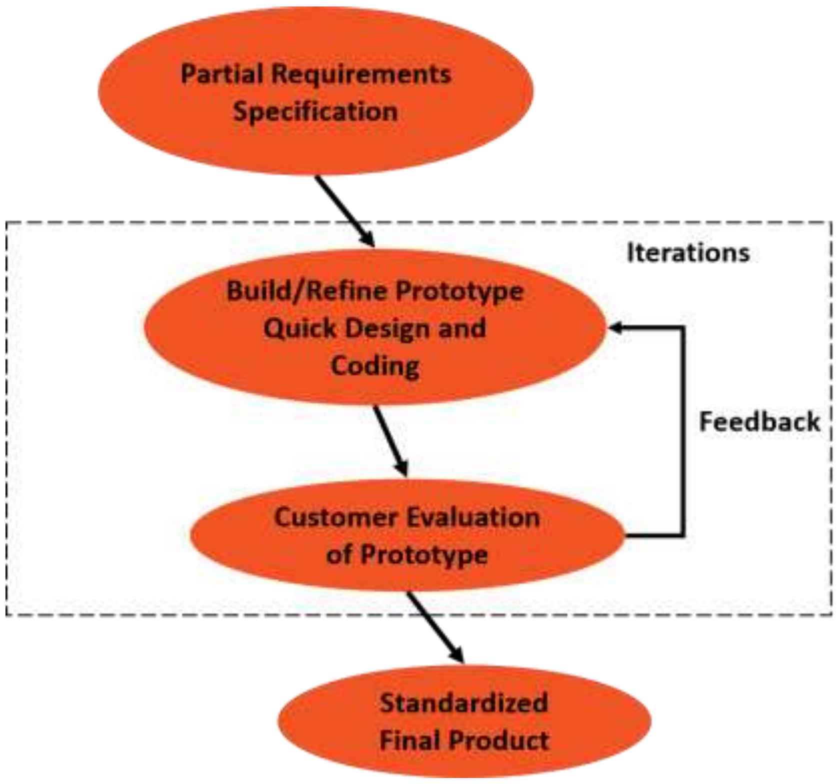
You can use the Waterfall model if -

- Requirements are very well known.
- Product definition is stable.
- Technology is well understood.
- New version of an existing product.
- Porting an existing product to a new platform.
- Large organization with structured cross-functional teams.
- Communication channels are well established within the organization and with the customer as well.

Evolutionary Prototyping Model

In software development using Evolutionary Prototyping model, the developers build a prototype during the requirements phase. The end users then evaluate the prototype and give feedback. The feedback can be corrections to the prototype or additional functionality. Based on the feedback, the developers further refine the prototype.

Thus, the product evolves through the Prototype -> Feedback -> Refined Prototype Cycles and hence the name Evolutionary Prototyping. When the user is satisfied with the functionality, and working of the product, the prototype code is brought up to the required standards for the final product delivery.


Partial Requirements Specification
Iterations
Build/Refine Prototype Quick Design and Coding
Feedback
Customer Evaluation of Prototype
Standardized Final Product

Evolutionary Prototyping Model – Strengths

The strengths or the advantages of an Evolutionary Prototyping model are-

- Customers / end users can visualize the system requirements as they are gathered looking at the prototype.
- Developers learn from customers and hence no ambiguities regarding domain or production environment.
- Allows flexible design and development.
- Interaction with the prototype stimulates the awareness of additionally needed functionality.
- Unexpected requirements and requirements changes are easily accommodated.
- Steady and visible signs of progress are produced.
- Delivery of an accurate and maintainable end-product.

Evolutionary Prototyping Model – Weaknesses

The weaknesses or disadvantages of the Evolutionary Prototyping model are as follows-

- Tendency to abandon structured development in the code-and-fix development, though it is not what is prescribed by the model.
- This model received bad reputation for the quick-and-dirty methods.
- Overall maintainability can possibly be overlooked.
- The customer can possibly ask for the delivery of the prototype as the final, not giving the opportunity for the developers to execute the final step i.e. standardization of the end-product.
- Project can continue forever (with continuous scope creep) and the management may not appreciate it.

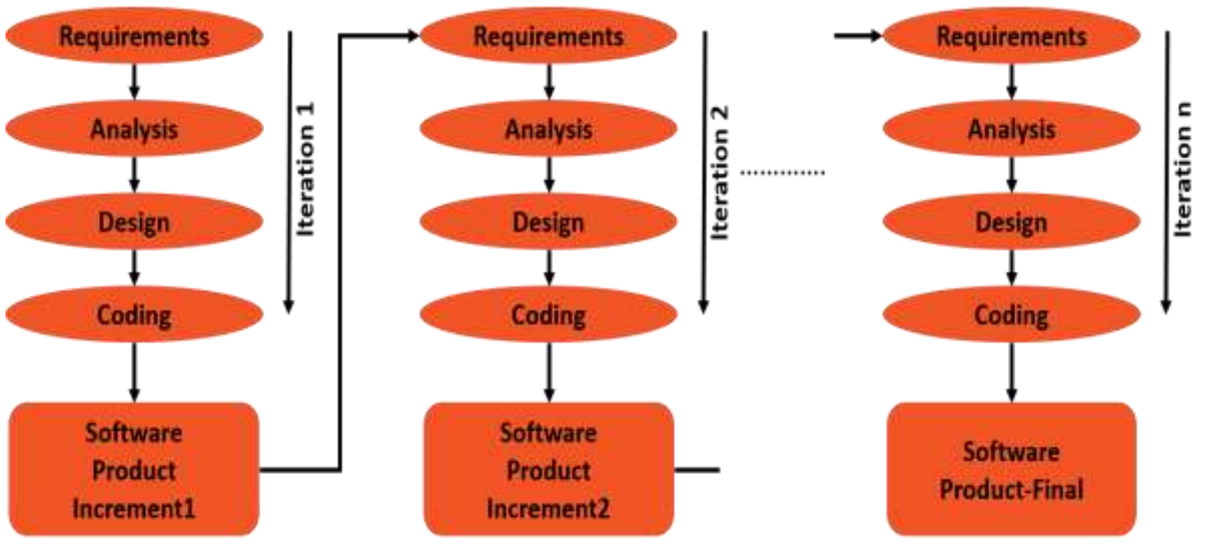
When to Use Evolutionary Prototyping Model?

You can use the Evolutionary Prototyping model-

- When requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- To develop user interfaces
- For short-lived demonstrations
- For new or original development
- For implementing a new technology

4. SDLC – Iterative Incremental Model

In an Iterative Incremental model, initially, a partial implementation of a total system is constructed so that it will be in a deliverable state. Increased functionality is added. Defects, if any, from the prior delivery are fixed and the working product is delivered. The process is repeated until the entire product development is completed. The repetitions of these processes are called iterations. At the end of every iteration, a product increment is delivered.



Iterative Incremental Model – Strengths

The advantages or strengths of Iterative Incremental model are-

- You can develop prioritized requirements first.
- Initial product delivery is faster.
- Customers gets important functionality early.
- Lowers initial delivery cost.
- Each release is a product increment, so that the customer will have a working product at hand all the time.
- Customer can provide feedback to each product increment, thus avoiding surprises at the end of development.
- Requirements changes can be easily accommodated.

Iterative Incremental Model – Weaknesses

The disadvantages of the Iterative Incremental model are-

- Requires effective planning of iterations.
- Requires efficient design to ensure inclusion of the required functionality and provision for changes later.
- Requires early definition of a complete and fully functional system to allow the definition of increments.
- Well-defined module interfaces are required, as some are developed long before others are developed.
- Total cost of the complete system is not lower.

When to Use Iterative Incremental Model?

Iterative Incremental model can be used when-

- Most of the requirements are known up-front but are expected to evolve over time.
- The requirements are prioritized.
- There is a need to get the basic functionality delivered fast.
- A project has lengthy development schedules.
- A project has new technology.
- The domain is new to the team.

5. SDLC – Rapid Application Development Model

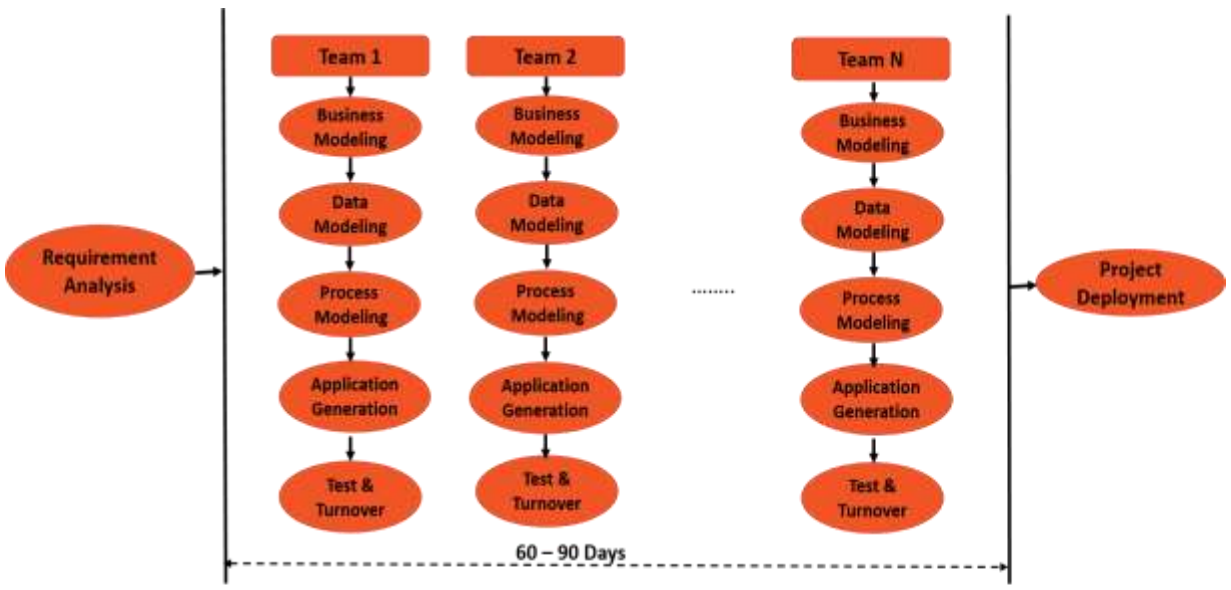
Rapid Application Development (RAD) model has the following phases-

- **Requirements Planning phase** - In the requirements planning phase, a workshop needs to be conducted to discuss business problems in a structured manner.

- **User Description phase** - In the User Description phase, automated tools are used to capture information from users.

- **Construction phase** - In the Construction phase, productivity tools, such as code generators, screen generators, etc. are used inside a time-box, with a "Do until Done" approach.

- **Cut Over phase** - In the Cut over phase, installation of the system, user acceptance testing and user training are performed.



Rapid Application Development Model – Strengths

The advantages or strengths of the Rapid Application Development model are as follows-

- Reduced cycle time and improved productivity with fewer team members would mean lower costs.

- Customer's involvement throughout the complete cycle minimizes the risk of not achieving customer satisfaction and business value.

- Focus moves to the code in a what-you-see-is-what-you-get mode (WYSIWYG). This brings clarity on what is being built is the right thing.

- Uses modelling concepts to capture information about business, data, and processes.

Rapid Application Development Model – Weaknesses

The disadvantages or strengths of Rapid Application Development model are as follows-

- Accelerated development process must give quick responses to the user.
- Risk of never achieving closure.
- Hard to use with legacy systems.
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to Use Rapid Application Development Model?

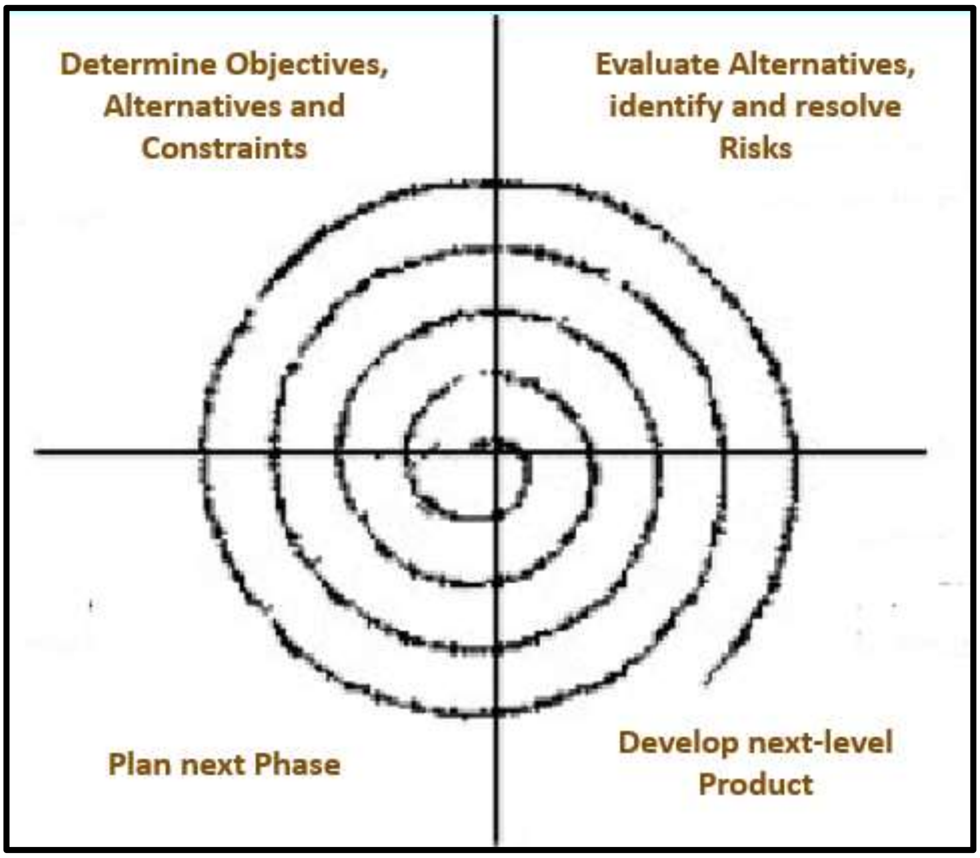
Rapid Application Development model can be used when-

- User can be involved throughout the life cycle.
- Project can be time-boxed.
- Functionality can be delivered in increments.

Though the strengths of Rapid Application Development model are appreciated, it is sparingly used in the industry.

6. SDLC – Spiral Model

Spiral model adds Risk Analysis and RAD prototyping to the Waterfall model. Each cycle involves the same sequence of steps as the Waterfall model.


Determine Objectives, Alternatives and Constraints
Evaluate Alternatives, identify and resolve Risks
Plan next Phase
Develop next-level Product

Spiral model has four quadrants. Let us discuss them in detail.

Quadrant 1: Determine objectives, alternatives and constraints

- **Objectives:** Functionality, performance, hardware/software interface, critical success factors, etc.
- **Alternatives:** Build, reuse, buy, sub-contract, etc.
- **Constraints:** Cost, schedule, interface, etc.

Quadrant 2: Evaluate alternatives, identify and resolve risks

- Study alternatives relative to the objectives and constraints that are determined.
- Identify risks such as lack of experience, new technology, tight schedules, etc.

- Resolve the identified risks evaluating their impact on the project, identifying the needed mitigation and contingency plans and implementing them. Risks always need to be monitored.

Quadrant 3: Develop next-level product

Typical activities include -

- Create a design
- Review design
- Develop code
- Inspect code
- Test product

Quadrant 4: Plan next phase

Typical activities include -

- Develop project plan
- Develop configuration management plan
- Develop a test plan
- Develop an installation plan

Spiral Model – Strengths

The advantages or strengths of the Spiral method are-

- Provides early indication of the risks, without involving much cost.
- Users can view the system early because of the rapid prototyping tools.
- Critical high-risk functions are developed first.
- The design does not have to be perfect.
- Users can be closely involved in all lifecycle steps.
- Early and frequent feedback from users.
- Cumulative costs assessed frequently.

Spiral Model – Weaknesses

The disadvantages or weaknesses of the Spiral method are-

- May be hard to define objectives, verifiable milestones that indicate readiness to proceed through the next iteration.
- Time spent in planning, resetting objectives, doing risk analysis and prototyping may be an overhead.
- Time spent for evaluating risks can be too large for small or low-risk projects.

- Spiral model is complex to understand for new team members.
- Risk assessment expertise is required.
- Spiral may continue indefinitely.
- Developers must be reassigned during non-development phase activities.

When to Use Spiral Model?

The Spiral model can be used when-

- Creation of a prototype is appropriate.
- Risk evaluation is important.
- A project is of medium to high-risk.
- Users are unsure of their needs.
- Requirements are complex.
- Product-line is new.
- Significant changes are expected during exploration.
- Long-term project commitment unwise because of potential business changes.

7. SDLC – Agile Methods

Agile Methods are based on the Agile manifesto and are adaptive in nature. Agile methods ensure-

- Team collaboration.
- Customer collaboration.
- Constant and continuous communication.
- Response to changes.
- Readiness of a working product.

Several Agile methods came into existence, promoting iterative and incremental development with time-boxed iterations. Though the Agile methods are adaptive, rules of the specific method cannot be by-passed and hence requires disciplined implementation.

Agile Methods – Strengths

The advantages or strengths of Agile method are-

- Early and frequent releases.
- Accommodation of changing requirements.
- Daily communication among the customer and developers.
- Projects built around motivated individuals.
- Self-organizing teams.
- Simplicity, focusing on what is immediately required.
- No building for future or overburdening the code.
- Regular reflection to adjust behavior to improve effectiveness.

Agile Methods – Weaknesses

The disadvantages or weaknesses of Spiral method are-

- Customer availability may not be possible.
- Teams should be experienced to follow the rules of the method.
- Appropriate planning is required to quickly decide on the functionality that needs to be delivered in an iteration.

- Team is expected to have estimation skills and negotiation skills.
- Team should have effective communication skills.
- New teams may not be able to organize themselves.
- Requires discipline to develop and deliver in time-boxed iterations.

- Design needs to be kept simple and maintainable, thus requiring effective design skills.

When to Use Agile methods?

The Agile methods can be used when-

- Application is time-critical.
- The scope is limited and less formal (scaling agile methods to larger projects is underway, with certain extensions to some of the agile methods).
- Organization employs disciplined methods.

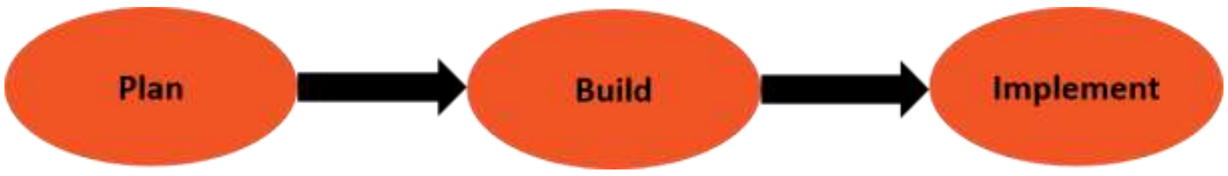
8. ASD – Evolution

The earlier SDLC models are more oriented to the practices of stability, predictability and decreasing returns. The industry, such as the Internet Platforms has been moving to increase return environments, unpredictable, nonlinear, and fast approaches.

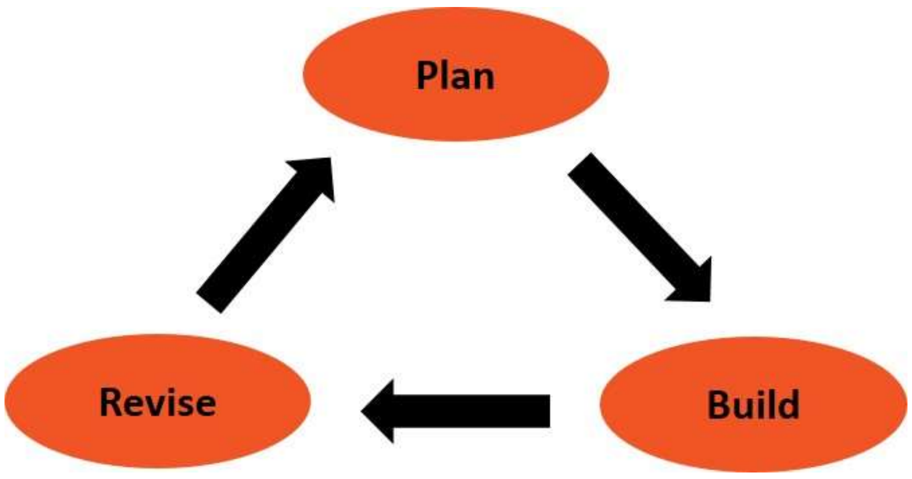
Adaptive Software Development (ASD) has evolved to address these issues. It focuses on emergence as the most important factor from the management’s perspective, to enhance the ability to manage product development.

In Jim Highsmith’s words, “Adaptive Software Development framework is based on years of experience with traditional Software Development methodologies, consulting on, practicing, and writing about Rapid Application Development (RAD) techniques and working with high-technology software companies on managing their product development practices”.

Waterfall model is found to be characterized by linearity and predictability, with meagre feedback. It can be viewed as a sequence of **Plan -> Build -> Implement**.



The Evolutionary Lifecycle models such as the Spiral model moved the Deterministic approach to the Adaptive one, with **Plan -> Build -> Revise Cycles**.



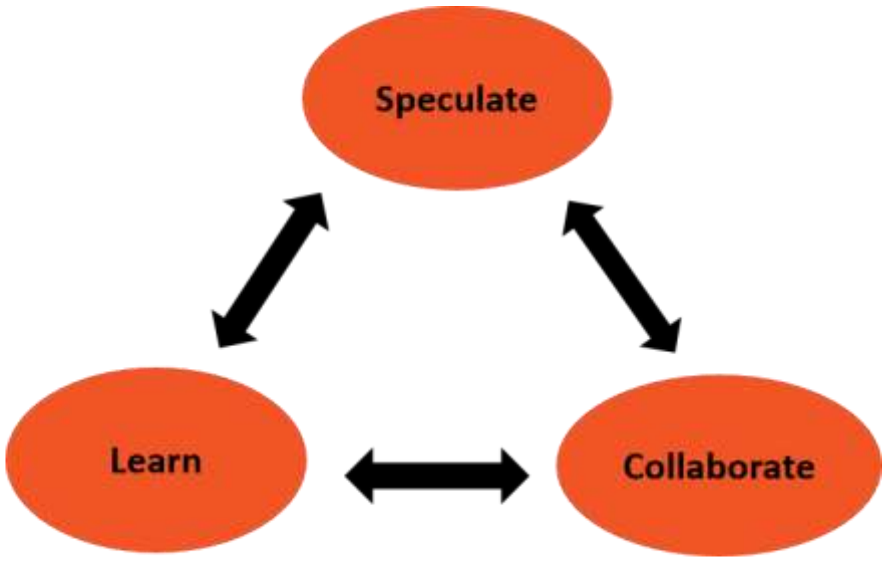
However, the practitioners’ mindset remained Deterministic with long-term predictability turning to short-term predictability. The practices of Evolutionary Lifecycle models such as RAD are found to be less Deterministic.

The Adaptive Life Cycle

The Adaptive model is built from a different point of view. Though cyclical like the Evolutionary model, the names of the phase reflect the unpredictable nature of increasingly complex systems.

Adaptive Development goes further than its evolutionary heritage in two key ways-

- It explicitly replaces Determinism with Emergence.
- It goes beyond a change in life cycle to a deeper change in management style.



The three phases in Adaptive Software Development Lifecycle are-

- **Speculate:** Speculate replaces the deterministic word planning, planning of product specifications or planning of project management tasks.

- **Collaborate:** Collaborate represents drawing a balance between
  - Managing in the traditional project management sense, and
  - Creating and maintaining the collaborative environment needed for emergence.

  Collaborative Activities build products, keeping up the pace of changes in the environment.

- **Learn:** Learn aims both, the developers and the customers, to use the results of each development cycle to learn the direction of the next.

9. ASD – Concepts

In this chapter, we will understand the various concepts of Adaptive Software Development.

Complex Adaptive Systems (CAS) Theory

Brian Arthur and his colleagues, at the Santa Fe institute, used the Complex Adaptive Systems (CAS) theory to revolutionize the understanding of Physics, Biology, Evolution, and Economics.

Brian Arthur culminated his more than two decades of trying to convince mainstream economists that their view, dominated by fundamental assumptions of decreasing returns, equilibrium, and deterministic dynamics, was no longer sufficient to understand reality. The new world is one of increasing returns, instability, and inability to determine cause and effect.

The two worlds differ in behavior, style, and culture. They call for-

- Different Management Techniques
- Different Strategies
- Different Understanding

Complex Software Development

With the scope of Software Applications being exploded, even the software development organizations are accruing similar contradictions as mentioned above.

- One World is represented by the Deterministic development, derived from management practices that are rooted with the basics of stability and predictability (which in Arthur's terms means decreasing returns)

- Second World is represented by the industries moving from decreasing to increasing return environments that are unpredictable, nonlinear and fast.

To address the issues of this second world, Jig Highsmith offered a framework, Adaptive Software Development that is different from the Deterministic Software Development.

The Adaptive Software Development focuses on addressing the complex systems-

- Adaptive Software Development for the development life cycle.

- Adaptive Management Techniques calling for a different mindset from that of traditional project management practices.

In this tutorial, you can understand both these implementations.

Adaptive Software Development (ASD) is based on two perspectives-

- Conceptual perspective based on the Complex Adaptive Systems (CAS) theory, as given in the first section of this chapter.

- Practical Perspective based on

    - Years of experience with Deterministic software development methodologies.

    - Consulting, practicing, and writing about Rapid Application Development (RAD) techniques; and working with high-technology software companies on managing their product development.

In this chapter, you will understand the conceptual perspective of Adaptive Software Development.

Complex Adaptive Systems (CAS) Concepts

Complex Adaptive Systems (CAS) theory has many concepts. Adaptive Software Development is based on two of these concepts-

- Emergence
- Complexity

Emergence

In complex software product-development projects, the outcomes are inherently unpredictable. However, successful products emerge from such environments all the time.

This can happen by Emergence, as illustrated in the Complex Adaptive Systems (CAS) theory. It can be understood by a simple example, flocking behavior of birds.

When you observe a flock of birds, you notice that-

- Each bird tries to
    - Maintain a minimum distance from other objects in the environment, including other birds.
    - Match velocities with birds in its neighborhood.
    - Move towards the perceived center of mass of birds in its neighborhood.

- There are no rules of behavior for the group. The only rules are about the behavior of individual birds.

- However, there exists an emergent behavior, the flocking of birds. When errant birds rush to catch up, the flock splits around obstacles and reforms on the other side.

This shows the requirement of the most difficult mental model changes in Adaptive Development – From ways of managing and organizing that individual freedom to the notion that a creative new order emerges unpredictably from spontaneous self-organization.

In addition to the development, emergence is the most important concept from the management perspective also.

Complexity

In the Software Development context, Complexity is about-

- The individuals of a team such as the developers, customers, vendors, competitors, and stockholders, their numbers and their speed.
- Size and technological complexity.

Adaptive Software Development Practices

Adaptive Software Development offers a different perspective on software management practices. In the sections below, you can understand the two important practices - Quality and RAD, both of which have ramifications for gathering requirements.

You can find the details of all the practices in the chapter, Adaptive Software Development Practices in this tutorial.

Quality

In a complex environment, the age-old practice of "Do it right the first time" does not work as you cannot predict what is right at the beginning. You need to have an aim to produce the right value. However, in complex environment, the combinations and permutations of value components like scope (features, performance, defect levels), schedule, and resources is so vast that there can never be an optimum value. Hence, the focus is to shift to deliver the best value in the competitive market.

RAD Practices

RAD Practices generally involve a combination of the following-

- Evolutionary Lifecycle
- Customer Focus Groups, JAD Sessions, Technical Reviews
- Time-boxed Project Management
- Continuous Software Engineering
- Dedicated Teams with war rooms

The RAD projects have an inherent adaptive, emergent flavor. Many IT organizations are against RAD. However, Microsoft and others have produced incredibly large and complex software using techniques comparable to RAD because it raises questions about their fundamental world view.

RAD practices and Microsoft process are both examples of Adaptive Development in action. Giving them a label (i.e., Adaptive Development) and realizing that there is a growing body of scientific knowledge (i.e., CAS theory) explains why they work. This should provide a basis for more extensive use of these practices.

10. ASD – Life Cycle

Adaptive Software Development has evolved from RAD practices. The team aspects also were added to these practices. Companies from New Zealand to Canada, for a wide range of project and product types, have used adaptive Software Development.

Jim Highsmith published Adaptive Software Development in 2000.

Adaptive Software Development practices provide ability to accommodate change and are adaptable in turbulent environments with products evolving with little planning and learning.

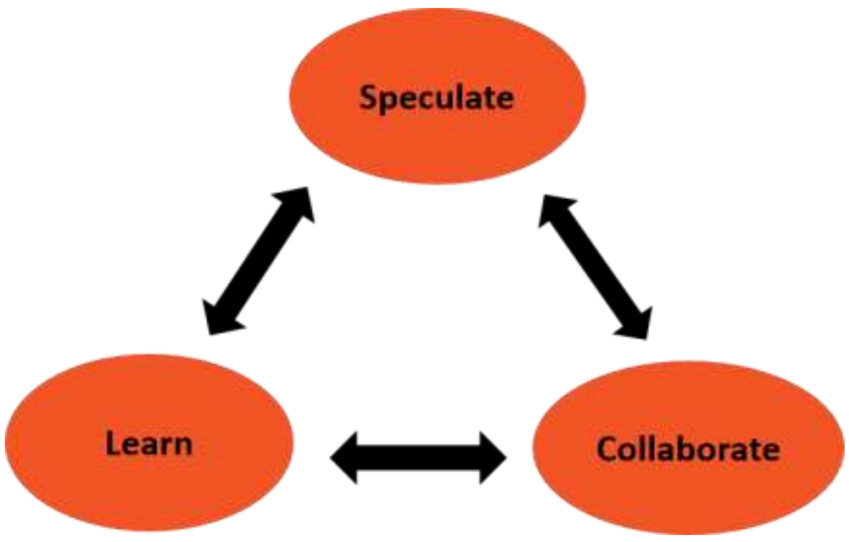
Phases of ASD Life Cycle

Adaptive Software Development is cyclical like the Evolutionary model, with the phase names reflecting the unpredictability in the complex systems. The phases in the Adaptive software development life cycle are-

- Speculate
- Collaborate
- Learn

These three phases reflect the dynamic nature of Adaptive Software Development. The Adaptive Development explicitly replaces Determinism with Emergence. It goes beyond a mere change in lifecycle to a deeper change in management style. Adaptive Software Development has a dynamic Speculate-Collaborate-Learn Lifecycle.

The Adaptive Software Development Lifecycle focuses on results, not tasks, and the results are identified as application features.



Speculate

The term plan is too deterministic and indicates a reasonably high degree of certainty about the desired result. The implicit and explicit goal of conformance to plan, restricts the manager's ability to steer the project in innovative directions.

In Adaptive Software Development, the term plan is replaced by the term speculate. While speculating, the team does not abandon planning, but it acknowledges the reality of uncertainty in complex problems. Speculate encourages exploration and experimentation. Iterations with short cycles are encouraged.

Collaborate

Complex applications are not built, they evolve. Complex applications require that a large volume of information be collected, analyzed, and applied to the problem. Turbulent environments have high rates of information flow. Hence, complex applications require that a large volume of information be collected, analyzed, and applied to the problem. This results in diverse Knowledge requirements that can only be handled by team collaboration.

Collaborate would require the ability to work jointly to produce results, share knowledge or make decisions.

In the context of project management, Collaboration portrays a balance between managing with traditional management techniques and creating and maintaining the collaborative environment needed for emergence.

Learn

The Learn part of the Lifecycle is vital for the success of the project. Team has to enhance their knowledge constantly, using practices such as-

- Technical Reviews
- Project Retrospectives
- Customer Focus Groups

Reviews should be done after each iteration. Both, the developers and customers examine their assumptions and use the results of each development cycle to learn the direction of the next. The team learns-

- About product changes
- More fundamental changes in underlying assumptions about how the products are being developed

The iterations need to be short, so that the team can learn from small rather than large mistakes.

Speculate - Collaborate - Learn Cycle as a Whole

As you observe from the Speculate-Collaborate-Learn cycle, given above, it is obvious that the three phases are nonlinear and overlap.

We observe the following from an Adaptive framework.

- It is difficult to Collaborate without Learning or to Learn without Collaborating.
- It is difficult to Speculate without Learning or to Learn without Speculating.
- It is difficult to Speculate without Collaborating or to Collaborate without Speculating.

11. ASD Lifecycle – Characteristics

Adaptive Software Development Lifecycle has six basic characteristics-

- Mission focused
- Feature based
- Iterative
- Time-boxed
- Risk driven
- Change tolerant

In this chapter, you will understand these six characteristics of Adaptive Software Development.

Mission-focused

For many projects, the overall mission that guides the team is well articulated, though the requirements may be uncertain at the beginning of the project. Mission statements act as guides that encourage exploration in the beginning but have a narrow focus over the course of a project. A mission provides boundaries rather than a fixed destination. Mission statements and the discussions that result in those statements provide direction and criteria for making critical project tradeoff decisions.

Without a clear mission and a constant mission refinement practice, iterative lifecycles become oscillating lifecycles, swinging back and forth with no progress in the development.

Feature-based

The Adaptive Software Development Lifecycle is based on application features and not on tasks. Features are the functionality that are developed during an iteration based on the customer's priorities.

Features can evolve over several iterations when the customers provide feedback.

The application features that provide direct results to the customer after implementation are primary. A customer-oriented document such as a user manual is also considered as a feature. The other documents such as the data model, even if defined as deliverables are always secondary.

Iterative

The Adaptive Software Development Lifecycle is iterative and focuses on frequent releases in order to obtain feedback, assimilate the resulting learning and setting the right direction for further development.

Time-boxed

In Adaptive Software Development Lifecycle, the iterations are time-boxed. However, one should remember that time-boxing in Adaptive Software Development is not about time deadlines. It should not be used to make the team work for long hours challenging a collaborative environment or for compromising on the quality of the deliverables.

In Adaptive Software Development, time-boxing is considered as a direction for focusing and forcing hard tradeoff decisions as and when required. In an uncertain environment, in which change rates are high, there needs to be a periodic forcing function such as a time-box to get the work finished.

Risk-driven

In Adaptive Software Development, the iterations are driven by identifying and evaluating the critical risks.

Change-tolerant

Adaptive Software Development is change-tolerant, viewing change as the ability to incorporate competitive advantage, but not as a problem for development.

12. ASD – Practices

The Adaptive Software Development practices are driven by a belief in continuous adaptation, with the lifecycle equipped to accepting continuous change as the norm.

Adaptive Software Development Lifecycle is dedicated to-

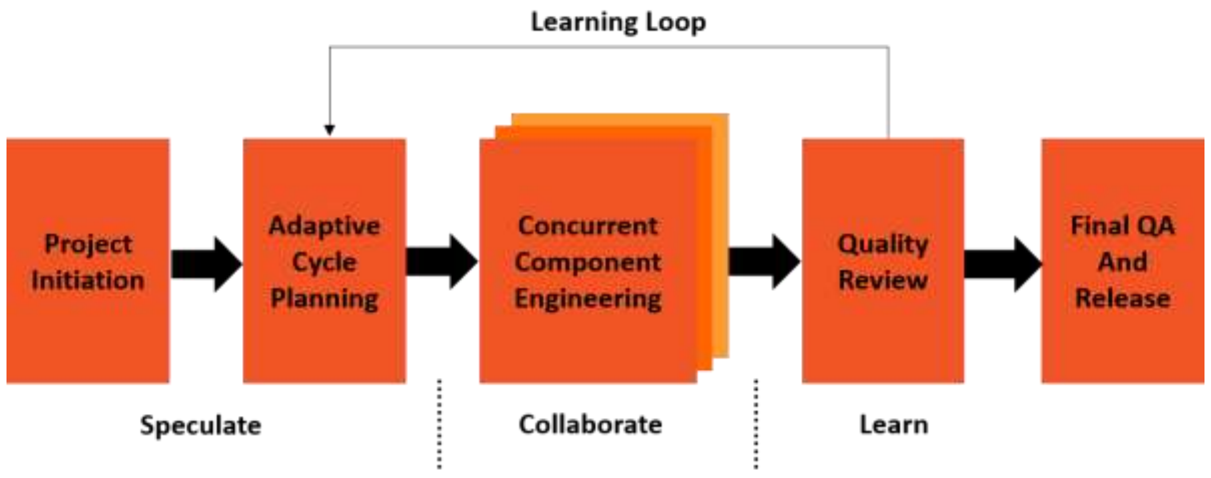
- Continuous learning
- Change orientation
- Re-evaluation
- Peering into an uncertain future
- Intense collaboration among developers, management, and customers

Adaptive SDLC

Adaptive Software Development combines RAD with Software Engineering Best Practices, such as-

- Project initiation.
- Adaptive cycle planning.
- Concurrent component engineering.
- Quality review.
- Final QA and release.

Adaptive Software Development practices can be illustrated as follows-



As illustrated above, Adaptive Software Development practices are spread across the three phases as follows-

- **Speculate –** Initiation and planning
  - Project Initiation
  - Establishing time-box for the entire project
  - Decide on the number of iterations and assign a time-box to each one
  - Develop a theme or objective for each of the iterations
  - Assign features to each iteration

- **Collaborate** – Concurrent feature development
  - Collaboration for distributed teams
  - Collaboration for smaller projects
  - Collaboration for larger projects

- **Learn** – Quality Review
  - Result quality from the customer's perspective
  - Result quality from a technical perspective
  - The functioning of the delivery team and the practices team members are utilizing
  - The project status

Speculate — Initiation and Planning

In Adaptive Software Development, the speculate phase has two activities-

- Initiation
- Planning

Speculate has five practices that can be executed repetitively during the initiation and planning phase. They are-

- Project initiation
- Establishing time-box for the entire project
- Decide on the number of iterations and assign a time-box to each one
- Develop a theme or objective for each of the iterations
- Assign features to each iteration

Project Initiation

Project Initiation involves-

- Setting the project's mission and objectives
- Understanding constraints
- Establishing the project organization
- Identifying and outlining requirements
- Making initial size and scope estimates
- Identifying key project risks

The project initiation data should be gathered in a preliminary JAD session, considering speed as the major aspect. Initiation can be completed in a concentrated two to five day effort for a small to medium sized projects, or two to three weeks effort for larger projects.

During the JAD sessions, requirements are gathered in enough detail to identify features and establish an overview of the object, data, or other architectural model.

Establishing Time-box for the Entire Project

The time-box for the entire project should be established, based on the scope, feature-set requirements, estimates, and resource availability that result from project initiation work.

As you know, Speculating does not abandon estimating, but it just means accepting that estimates can go wrong.

Iterations and Time-box

Decide on the number of iterations and the individual iteration lengths based on the overall project scope and the degree of uncertainty.

For a small to medium sized application-

- Iterations usually vary from four to eight weeks.
- Some projects work best with two-week iterations.
- Some projects might require more than eight weeks.

Choose the time, based on what works for you. Once you decide on the number of iterations and the lengths of each of the iterations, assign a schedule to each of the iterations.

Develop a Theme or Objective

The team members should develop a theme or objective for each iteration. This is something similar to the Sprint Goal in Scrum. Each iteration should deliver a set of features that can demonstrate the product functionality making the product visible to the customer to enable review and feedback.

Within the iterations, the builds should deliver working features on a preferably daily basis enabling integration process and making the product visible to the development team.

Testing should be an ongoing, integral part of the feature development. It should not be delayed until the end of the project.

Assign Features

Developers and customers should together assign features to each iteration. The most important criteria for this feature assignment is that every iteration must deliver a visible set of features with considerable functionality to the customer.

During the assignment of features to the iterations-

- Development team should come up with the feature estimates, risks, and dependencies and provide them to the customer.

- Customers should decide on feature prioritization, using the information provided by the development team.

Thus iteration planning is feature-based and done as a team with developers and customers. Experience has shown that this type of planning provides better understanding of the project than a task-based planning by the project manager. Further, feature-based planning reflects the uniqueness of each project.

Collaborate — Concurrent Feature Development

During the Collaborate phase, the focus is on the development. The Collaborate phase has two activities-

- The Development team collaborate and deliver working software.

- The project managers facilitate collaboration and concurrent development activities.

Collaboration is an act of shared creation that encompasses the development team, the customers and the managers. Shared creation is fostered by trust and respect.

Teams should collaborate on-

- Technical problems
- Business requirements
- Rapid decision making

Following are the practices relevant to the Collaborate phase in Adaptive Software Development-

- Collaboration for distributed teams
- Collaboration for smaller projects
- Collaboration for larger projects

Collaboration for Distributed Teams

In the projects involving distributed teams, the following should be considered-

- Varying alliance partners
- Broad-based knowledge
- The way people interact
- The way they manage interdependencies

Collaboration for Smaller Projects

In the smaller projects, when team members are working in physical proximity, Collaboration with informal hallway chats and whiteboard scribbling should be encouraged, as this is found to be effective.

Collaboration for Larger Projects

Larger projects require additional practices, collaboration tools, and project manager interaction and should be arranged on the contextual basis.

Learn - Quality Review

Adaptive Software Development encourages the concept of ‘Experiment and Learn’.

Learning from the mistakes and experimentation requires that the team members share partially completed code and artifacts early, in order to-

- Find mistakes
- Learn from them
- Reduce rework by finding small problems before they become large ones

At the end of each development iteration, there are four general categories of things to learn-

- Result quality from the customer's perspective
- Result quality from a technical perspective
- The functioning of the delivery team and the practices team
- The project status

Result Quality from the Customer's Perspective

In the Adaptive Software Development projects, getting feedback from the customers is the first priority. The recommended practice for this is a customer focus group. These sessions are designed to explore a working model of the application and record customer change requests.

Customer focus group sessions are facilitated sessions, similar to jad sessions, but rather than generating requirements or defining project plans, they are designed to review the application itself. The customers provide feedback on the working software resulting from an iteration.

Result Quality from a Technical Perspective

In the Adaptive Software Development projects, periodic review of technical artifacts should be given importance. Code Reviews should be done on a continuous basis. Reviews of other technical artifacts, such as technical architecture can be conducted weekly or at the end of an iteration.

In Adaptive Software Development projects, the team should monitor its own performance periodically. Retrospectives encourage the teams to learn about themselves and their work, together as a team.

Iteration-end retrospectives facilitate periodic team performance self-review such as-

- Determine what is not working.
- What the Team needs to do more.
- What the Team needs to do less.

The Project Status

The Project status review helps in planning further work. In the adaptive software development projects, determining the project status is feature-based approach, the end of each iteration marked by completed features resulting in working software.

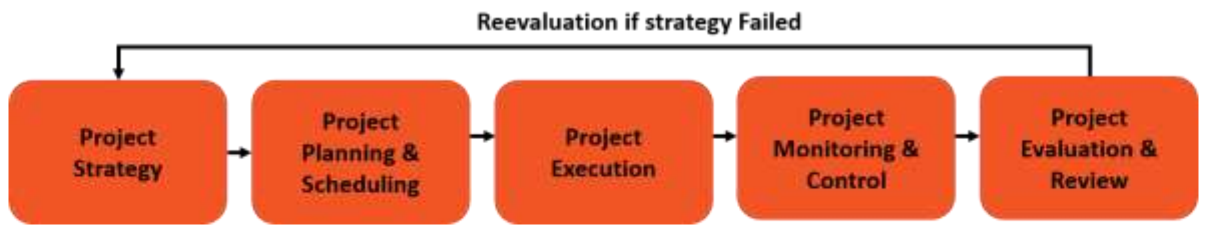
The Project Status review should include-

- Where is the project?
- Where is the project versus the plans?
- Where should the project be?

As the plans in the Adaptive Software Development projects are speculative, more than the question 2 above, question 3 is important. That is, the project team and the customers need to continuously ask themselves, "What have we learned so far, and does it change our perspective on where we need to go?"

13. ASD – Adaptive Management

A flowchart of the Traditional software management is shown below.


Reevaluation if strategy Failed
Project Strategy
Project Planning & Scheduling
Project Execution
Project Monitoring & Control
Project Evaluation & Review

Traditional software management has been characterized by the term command-control.

Many organizations are steeped in a tradition of optimization, efficiency, predictability, control, rigor and process improvement. However, the emerging information age economy requires adaptability, speed, collaboration, improvisation, flexibility, innovation, and suppleness.

Harvard business review and management books have come up with the terms such as empowerment, participative management, learning organization, human-centered management, etc., but none of these are being put into managing modern organizations.

In the context of Adaptive Software Development, the gap looks much wider and there is a necessity to consider the Adaptive management techniques that have been proven successful in other fields.

Adaptive Management

Adaptive management has proven successful in the environments where the resource managers worked together with stakeholders and scientists as a team, with the following goals-

- To learn how managed systems respond to human interventions.
- To improve resource policies and practices in future.

The principle behind adaptive management is that many resource management activities are experiments as their outcomes cannot be reliably predicted beforehand. These experiments are then used as learning opportunities for the improvements in the future.

Adaptive management is intended to increase the ability to respond timely in the face of new information and in a setting of varied stakeholder objectives and preferences. It encourages stakeholders to bound disputes and discuss them in an orderly fashion while the environmental uncertainties are being investigated and better understood.

Adaptive management helps the stakeholders, the managers and other decision makers recognize the limits of knowledge and the need to act on imperfect information.

Adaptive management helps to change the decisions made by making it clear that-

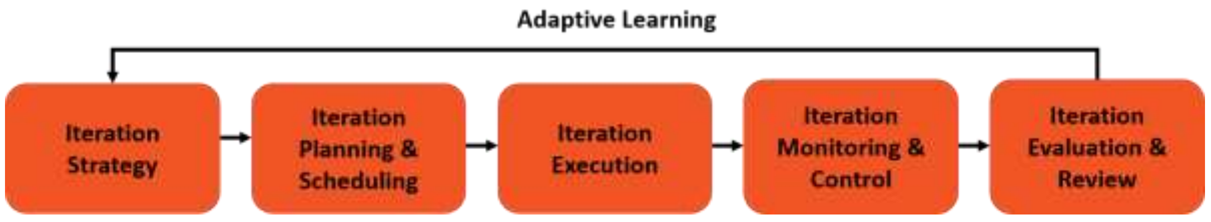
- The decisions are provisional.
- A management's decision need not always be right.
- Modifications are expected.

There are two types of Adaptive management approaches-

- Passive Adaptive Management.
- Active Adaptive Management.

Passive Adaptive Management

Adaptive management aims to enhance the scientific knowledge and thereby reduce uncertainties.

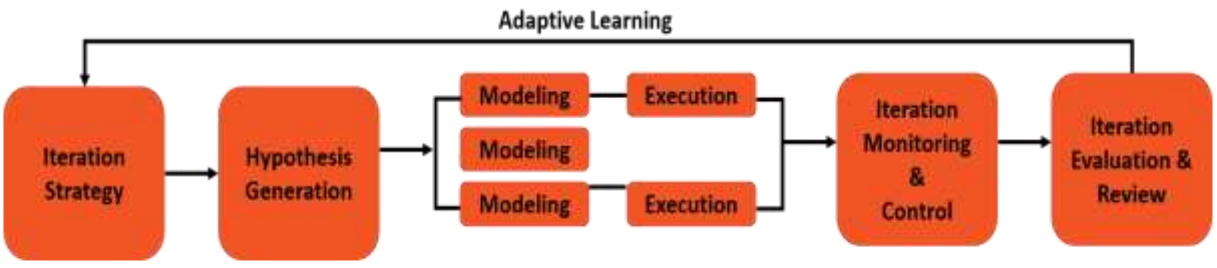

Adaptive Learning
Iteration Strategy
Iteration Planning & Scheduling
Iteration Execution
Iteration Monitoring & Control
Iteration Evaluation & Review

Within Passive Adaptive management, a single preferred course of action, based on existing information and understanding, is selected. The outcomes of management actions are monitored, and subsequent decisions are adjusted based on the outcomes.

This approach contributes to the learning and effective management. However, it is limited in its ability to enhance scientific and management capabilities for conditions that go beyond the course of action selected.

Active Adaptive Management

An Active Adaptive management approach reviews the information before management actions are taken.



A range of competing, alternative system models of ecosystem and related responses (e.g. demographic changes; recreational uses), rather than a single model, is then developed. Management options are chosen based on the evaluations of these alternative models.

Leadership-Collaboration Management

Adaptive management is what is best suited for Adaptive Software Development. The approach requires resource managers, i.e. the managers who can work with people, allow human-interventions, and create an amicable environment.

In software development, the leaders often take up these responsibilities. We need leaders more than the commanders. The leaders are collaborators and work alongside with the team. Collaborative-Leadership is the most sought after practice in Adaptive development.

The leaders have the following qualities-

- Grasp and set the direction.
- Influence people involved and provide guidance.
- Collaborate, facilitate and macro-manage the team.
- Provide direction.
- Create environments where talented people can be innovative, creative, and make effective decisions.
- Understand that occasionally they need to command, but that is not their predominant style.