TheStudentCircle.com
Circle of your Dreams

www.thestudentcircle.com

www.facebook.com/thestudentcircle27

Adaptive Software Development - Study Notes

Adaptive Software Development Study Notes

Adaptive Software Development is a move towards adaptive practices, leaving the deterministic practices with regards to complex systems and complex environments. Adaptive Software Development focuses on collaboration and learning as a method 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 searching for ways to improve performance and to moderate burnout, particularly as the projects they embrace get bigger and the teams become progressively appropriated.

Prerequisites

Before you begin continuing with this study notes, we are expecting that you are already aware about the basics of Software Development Life Cycle(SDLC). If you are not well aware of these concepts, then we will suggest you to go through our short study notes on SDLC.

Adaptive S/W Development - Introduction

What Is Agile?

In literary terms, the word “agile” means somebody who can move rapidly and effectively or somebody who can think and act rapidly and clearly. In business, “agile” is utilized for describing methods of planning and accomplishing work wherein it is understood that making changes as needed is an significant role of the job. Business “agility” implies that a company is consistently in a position to take account of the market changes.

In software development, the expression “agile” is adapted to signify “the ability to respond to changes – changes from Requirements, Technology and People.”

Agile Manifesto

The Agile Manifesto was published by a group of software engineers in 2001, highlighting the significance of the development team, accommodating changing requirements and client involvement.

The Agile Manifesto is –

We are revealing better methods of developing software by doing it and helping other people do it. Through this work, we have come to value –

- Individuals and cooperations over processes and tools.
- Working software over comprehensive documentation.
- Client collaboration over agreement negotiation.
- Responding to change over following a plan.

That is, while there is an incentive in the things 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 entire team with multi-discipline, cross-functional teams that are engaged and selforganizing.
- It encourages shared responsibility and accountability.
- Facilitates effective communication and consistent collaboration.
- The entire team approach avoids delays and wait times.
- Frequent and continuous deliveries ensure quick feedback that in in turn enable the team align to the prerequisites.
- Collaboration encourages consolidating different perspectives timely in execution, bug fixes and accommodating changes.

- Progress is constant, sustainable, and predictable emphasizing transparency.

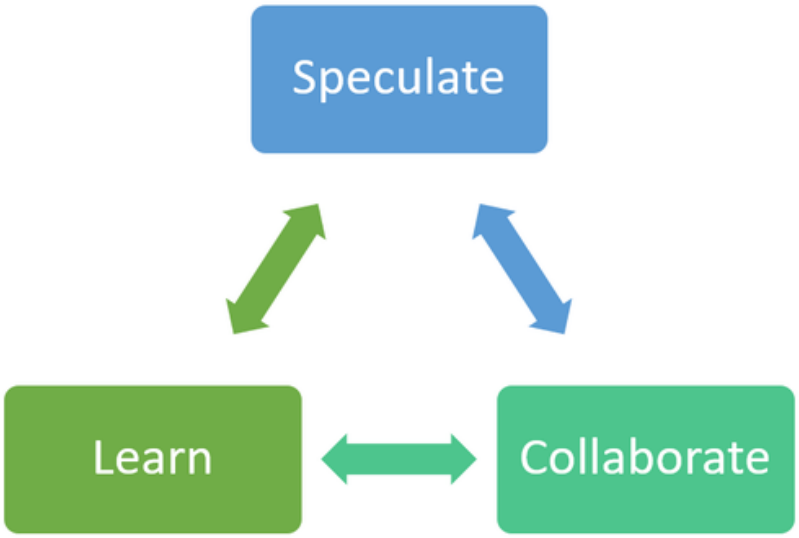
Agile Methodologies

Early implementations of Agile methods incorporate 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 study notes, we will get familiar with 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 cooperation and learning as a method 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 incorporate adaptive methodologies 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."

SDLC Models - Evolution

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

In a Software Development Life Cycle, the activities are performed in five stages −

- **Requirements Gathering** − Requirements for a software to be developed are accumulated. These requirements will be in a language that is comprehended by the client/user. Domain specific terminology is recommended.
- **Analysis** − The gathered prerequisites are analyzed from implementation perspective and the software specifications are written to cover both, the functional requirements and the non-functional requirements.
- **Design** − This stage involves arriving at the software architecture and execution specifics based on technology chosen for development.
- **Development** − In this stage, the code is developed, unit tested, integrated, integration tested and the build is delivered.
- **Testing** − Functional testing of the built software is done in this stage. This additionally incorporates the testing of non-functional requirements.

There are two ways to performing these activities −

- **Prescriptive** − The SDLC models that will give you methods of performing the activities in a endorsed way as defined by the framework.
- **Adaptive** − The SDLC models that will give you flexibility in performing the activities, with specific guidelines that need to be followed. The agile methods mostly follow this approach, with each one having its standards. In any case, following an adaptive or agile approach does not mean that the software is developed without following any order. This would lead to a confusion.

You need to understand that we can't say that a particular SDLC model is fortunate or unfortunate. Every one of them has its own strengths and weaknesses and thus are suitable in specific settings.

When you choose an SDLC model for your project, you need to understand below points −

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

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

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 every one of them and in what contexts they are reasonable.

SDLC - Waterfall Model

The Waterfall model is a classic SDLC model that is broadly known, understood and normally used. It was presented by Royce in 1970 and is still being followed as a typical approach for software development in different organizations across the industry.

In Waterfall model, each lifecycle stage can begin only after the earlier lifecycle stage is complete. Consequently, it is a linear model with no feedback loops.

Waterfall Model – Advantages

The advantages of the Waterfall model are following –

- Straightforward, easy to use.
- Gives structure to inexperienced development team.
- Milestones are well understood.
- Sets requirements stability.
- Perfect for management control (planning, monitoring, reporting).
- Works well when quality is higher priority than cost or schedule.

Waterfall Model – Disadvantages

The weaknesses or the disadvantages of the Waterfall model are following –

- Idealised – It does not match reality well.
- Unrealistic – can't expect accurate requirements early in the project.
- Doesn't reflect iterative nature of exploratory development that is more common.
- Difficult and expensive to make changes.
- Software is delivered only at the finish of the project. Because of this –

  - Delays discovery of genuine defects.
  - Possibility of delivery of out of date requirements.
- Significant management overhead, which can be costly for small teams and projects.
- Requires experienced assets at every phase – analysts, designers, developers, testers.
- Testing begins simply after the development is finished and the testers are not engaged in any of the earlier phases.
- The expertize of the cross-functional teams is not shared as each phase is executed in storehouses.

Usage of Waterfall Model?

You can utilize 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 inside the organization and with the customer as well.

Transformative Prototyping Model

In software development using Transformative Prototyping model, the engineers assemble a prototype during the requirements phase. The end clients then evaluate the prototype and give feedback. The feedback can be amendments to the prototype or additional functionality. Based on the feedback, the developers further refine the prototype.

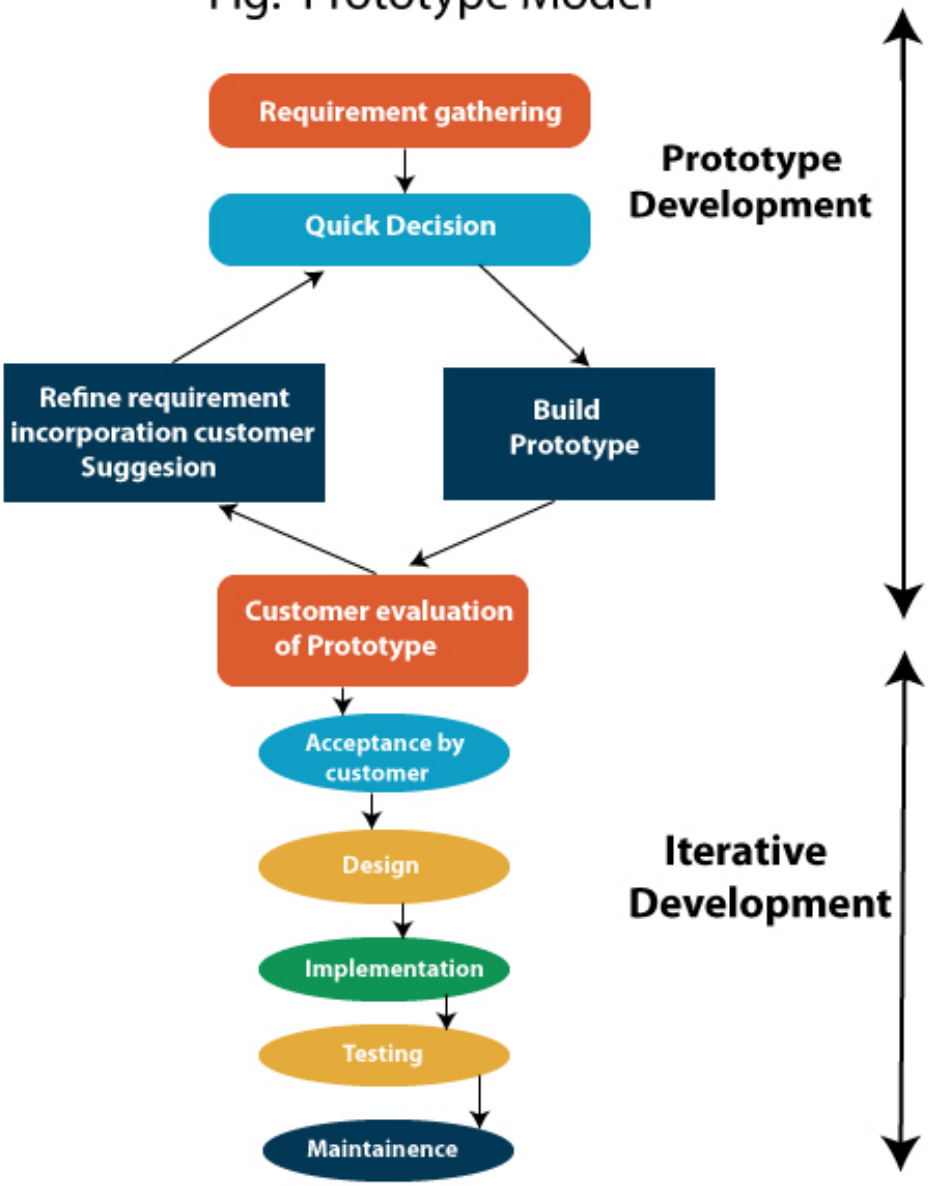
Transformative Prototyping Model – Strengths

The strengths or the advantages of an Transformative 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.

Fig: Prototype Model



Transformative Prototyping Model – Weaknesses

The weaknesses or disadvantages of the Transformative 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.

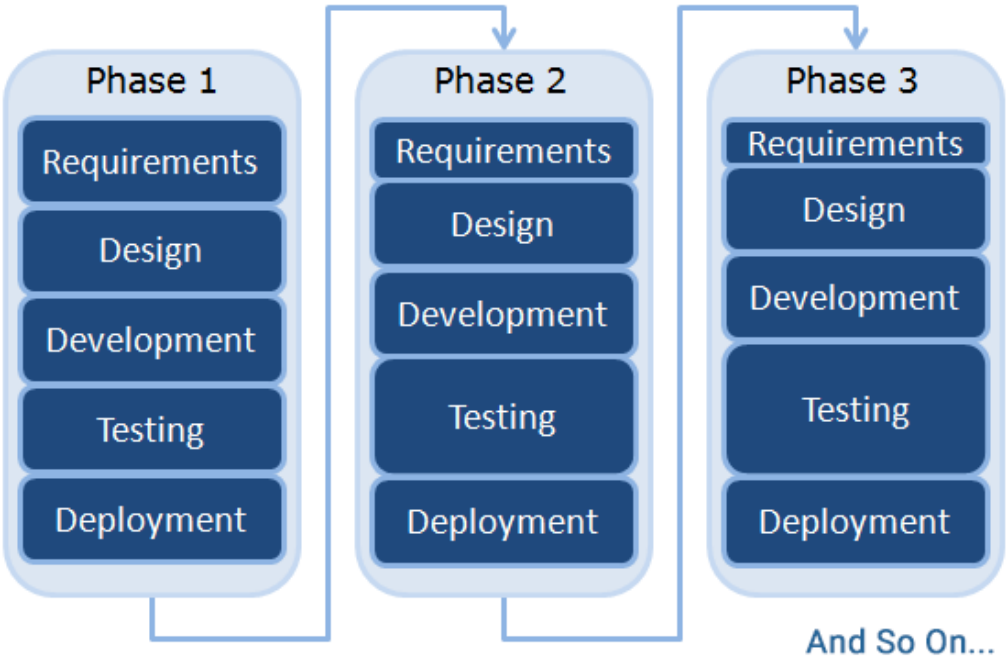
When to Use Transformative Prototyping Model?

You can use the Transformative 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

SDLC - Iterative Incremental Model

In an Iterative Incremental model, at first, a partial implementation of a complete system is built so that it will be in a deliverable state. Expanded functionality is included. Defects, if any, from the prior delivery are fixed and the working product is delivered. The procedure is repeated until the entire product development is finished. 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 following –

- You can develop prioritized requirements first.
- Initial product delivery is faster.
- Clients 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.
- Client 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?

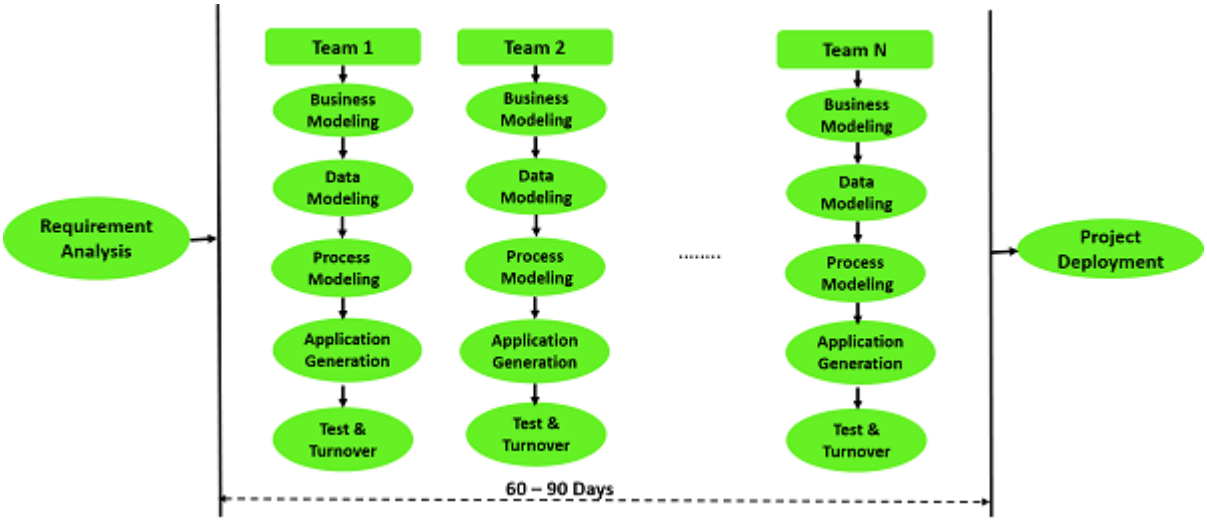
Iterative Incremental model can be used when –

- Most of the prerequisites are known up-front but are expected to evolve over time.
- The prerequisites 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.

SDLC - Rapid Application Development Model

Rapid Application Development (RAD) model has the following stages −

- **Requirements Planning stage** − In the requirements planning stage, a workshop needs to be conducted to talk business problems in a organized manner.
- **User Description stage** − In the User Description stage, automated tools are utilized to capture data from users.
- **Development stage** − In the Development stage, productivity tools, such as code generators, screen generators, etc. are used inside a time-box, with a "Do until Done" approach.
- **Cut Over stage** − In the Cut over stage, 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 duration and improved productivity with fewer colleagues would mean lower costs.
- Client's involvement throughout the complete cycle minimizes the risk of not achieving client satisfaction and business value.

  Client's contribution all through the total cycle limits the danger of not accomplishing consumer loyalty and business esteem.

- 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 conclusion.
- Difficult to use with legacy systems.
- Developers and clients must be focused on rapid-fire activities in a condensed time frame.

When to Use Rapid Application Development Model?

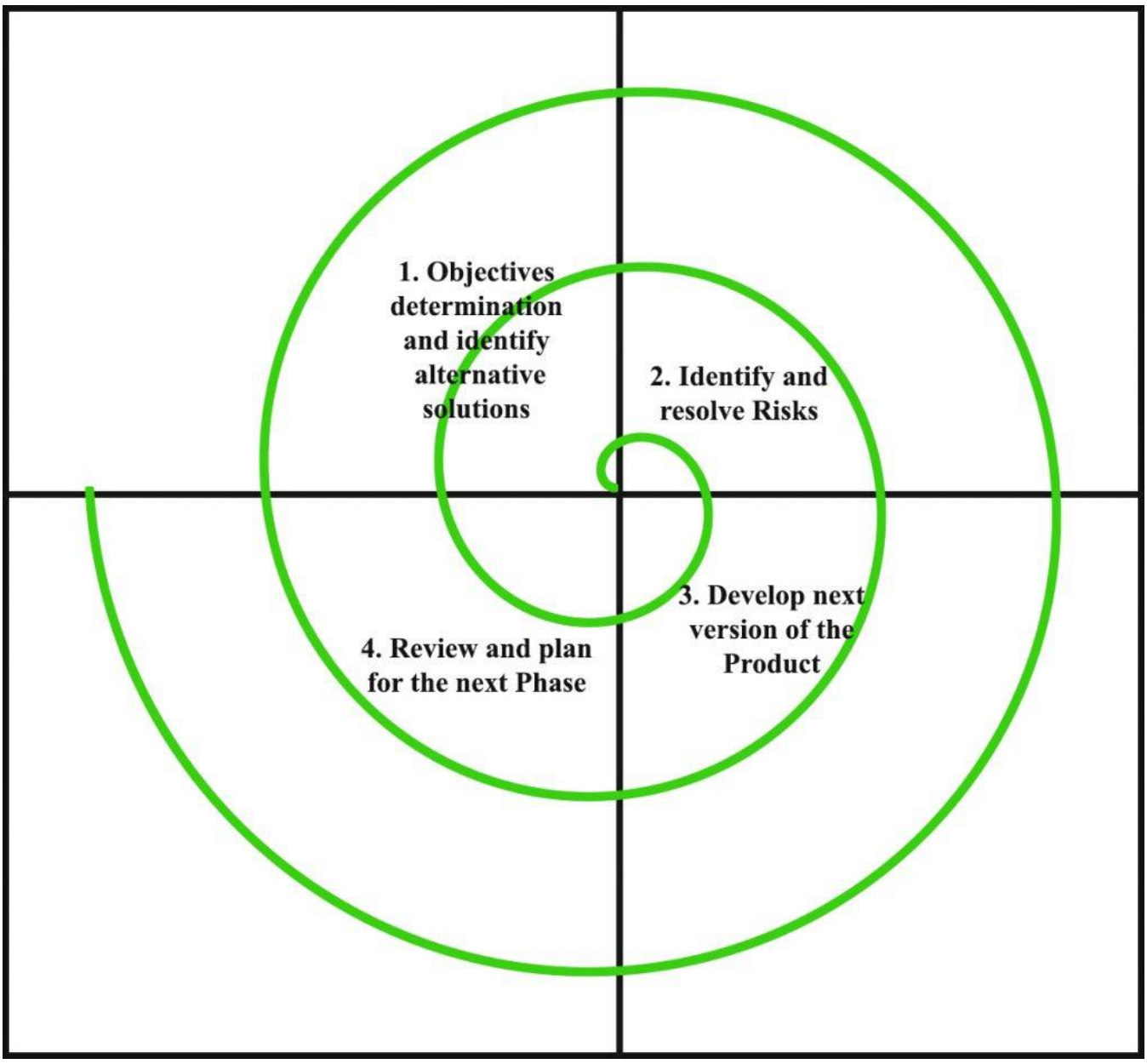
Rapid Application Development model can be used when −

- 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 utilized in the business.

SDLC - Spiral Model

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



Spiral model has four quadrants. Let us talk about them in detail.

Quadrant 1 - Determine goals, alternatives and constraints

- **Goals** – Functionality, performance, hardware/software interface, critical achievement factors, etc.
- **Alternatives** – Build, reuse, purchase, sub-contract, etc.
- **Constraints** – Cost, plan, interface, etc.

Quadrant 2 - Evaluate options, recognize and resolve risks

- Study choices relative to the objectives and constraints that are determined.
- Identify risks such as lack of experience, new innovation, tight calendars, etc.
- Resolve the recognized risks evaluating their impact on the project, identifying the required mitigation and contingency plans and implementing them. Risks consistently should be monitored.

Quadrant 3 - Develop next-level product

Typical activities incorporate –

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

Quadrant 4 - Plan next stage

Typical activities incorporate –

- 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 including a lot of cost.
- Users can view the system early because of the rapid prototyping tools.
- Critical high-risk functions are developed first.
- The design doesn't have to be perfect.
- Users can be firmly involved in all lifecycle steps.
- Early and frequent criticism from users.
- Cumulative costs assessed frequently.

Spiral Model – Weaknesses

The disadvantages or weaknesses of the Spiral strategy are –

- May be difficult 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 in arranging risks can be too large for small or low-risk projects.
- Spiral model is unpredictable to understand for new team members.
- Risk assessment expertise is required.
- Spiral may continue uncertainly.
- Developers must be reassigned during non-development stage activities.

When to Use Spiral Model?

The Spiral model can be utilized when −

- Creation of a prototype is proper.
- Risk assessment is significant.
- A project is of medium to high-risk.
- Users are unsure of their needs.
- Requirements are complex.
- Product-line is new.
- Significant changes are normal during investigation.
- Long-term project commitment unwise because of potential business changes.

SDLC - Agile Methods

Agile Methods depend on the Agile manifesto and are versatile in nature. Agile methods ensure −

- Team collaboration.
- Client collaboration.
- Consistent and continuous communication.
- Reaction to changes.
- Readiness of a working product.

Several Agile methods came into existence, promoting iterative and incremental development with time-boxed iterations. Despite the Agile methods are adaptive, rules of the specific method can't be by-passed and henceforth requires disciplined implementation.

Agile Methods – Strengths

The advantages or strengths of Agile method are −

- Early and frequent releases.
- Accommodation of changing prerequisites.
- Daily communication among the client and developers.
- Projects built around motivated people.
- 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 viability.

Agile Methods – Weaknesses

The disadvantages or weaknesses of Spiral method are −

- Client availability may not be possible.
- Teams should be experienced to follow the rules of the method.
- Appropriate planning is required to rapidly 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 straightforward and maintainable, thus requiring effective design skills.

When to Use Agile methods?

The Agile methods can be utilized when –

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

Adaptive S/W Development - Evolution

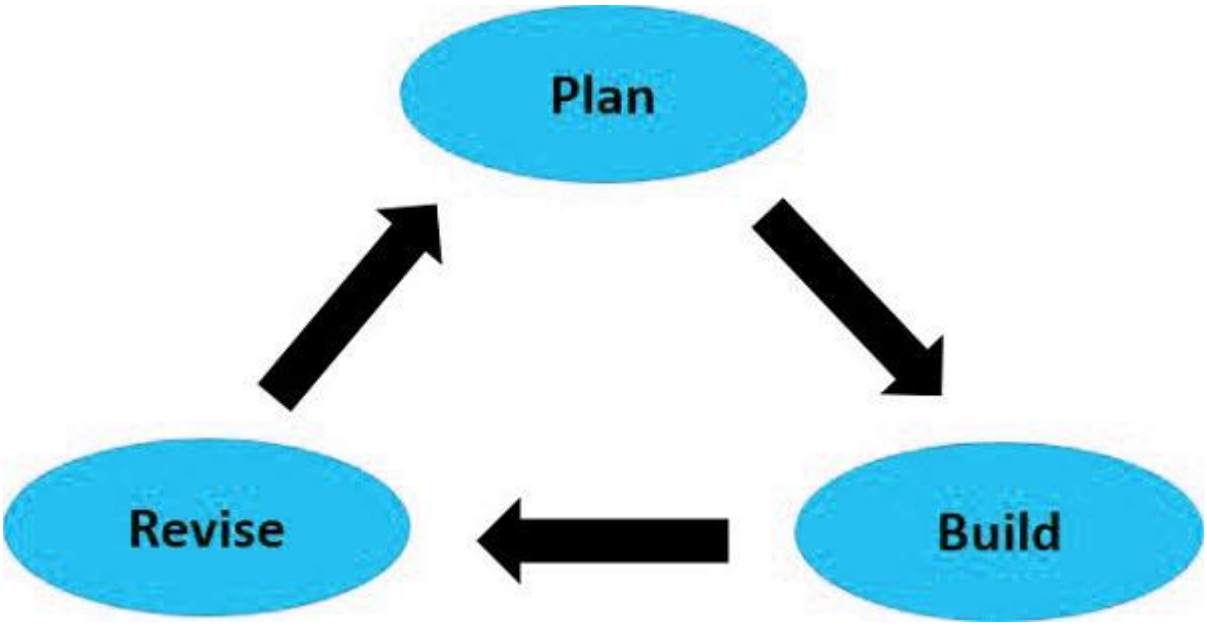
The earlier SDLC models are more oriented to the practices of stability, consistency and decreasing returns. The business, such as the Internet Platforms has been moving to increase return environments, unpredictable, nonlinear, and quick methodologies.

Adaptive Software Development (ASD) has advanced to address these issues. It focuses on emergence as the most significant factor from the management's perspective, to enhance the capacity 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) strategies and working with high-innovation software organizations on managing their product development practices".

Waterfall model is found to be described by linearity and consistency, 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 way to the Adaptive one, with **Plan** → **Build** → **Revise Cycles**.



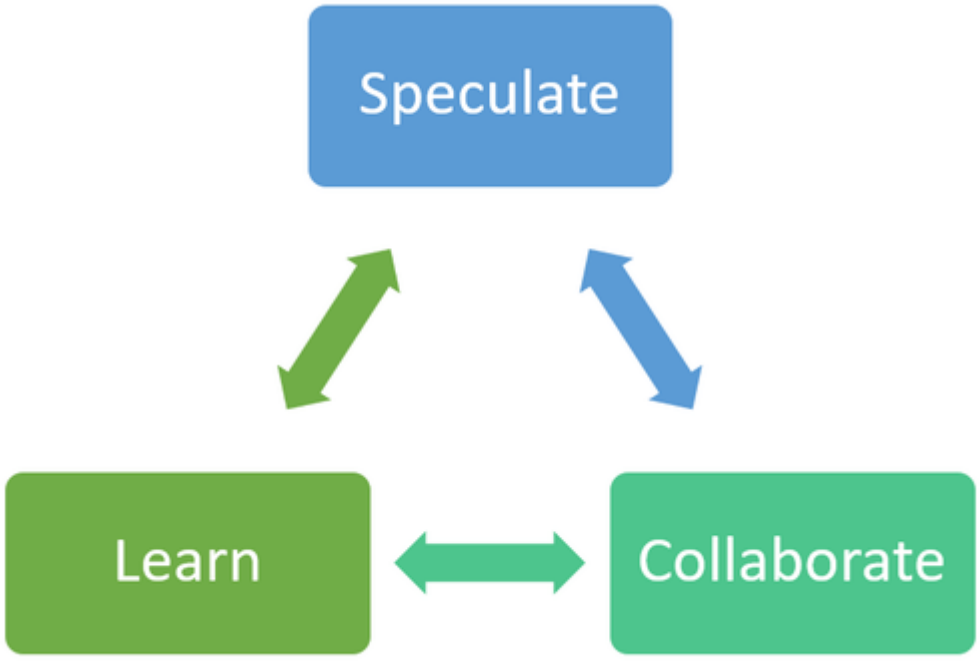
However, the practitioners' mindset remained Deterministic with long-term consistency turning to short-term consistency. The practices of Evolutionary Lifecycle models such as RAD are found to be less Deterministic.

The Adaptive Life Cycle

The Adaptive model is built from a alternate point of view. Despite the fact that cyclical like the Evolutionary model, the names of the stage reflect the unpredictable idea of progressively complex systems.

Adaptive Development goes further than its evolutionary heritage in two key manners −

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



The three stages in Adaptive Software Development Lifecycle are −

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

Collective Activities assemble items, keeping up the pace of changes in the earth.

- **Speculate** − Speculate replaces the deterministic word planning, planning of product specifications or planning of project management assignments.
- **Collaborate** − Collaborate speaks to drawing a balance between
    - Managing in the traditional project management sense, and
    - Creating and maintaining the collaborative environment needed for development.
- **Learn** − Learn aims both, the developers and the clients, to utilize the results of each development cycle to learn the direction of the next.

Adaptive S/W Development - Concepts

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

Complex Adaptive Systems (CAS) Theory

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

Brian Arthur finished his more than two years of trying to convince mainstream economists that their view, ruled by fundamental assumptions of decreasing returns, equilibrium, and deterministic dynamics, was no longer sufficient to understand reality. The new world is one of expanding returns, instability, and failure 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 detonated, even the software development organizations are collecting similar contradictions as mentioned above.

- One World is represented by the Deterministic development, derived from management practices that are established with the basics of stability and consistency (which in Arthur's terms means decreasing returns)
- Second World is represented by the industries moving from decreasing to expanding return environments that are unpredictable, nonlinear and quick.

To address the issues of this second world, Jig Highsmith offered a framework, Adaptive Software Development that is unique in relation to the Deterministic Software Development.

The Adaptive Software Development focuses on tending the complex systems –

- Adaptive Software Development for the development life cycle.
- Adaptive Management Techniques requiring an alternate mindset from that of traditional project management practices.

In this study notes, you can understand both these implementations.

Adaptive Software Development (ASD) is based on two points of view –

- Conceptual perspective dependent on the Complex Adaptive Systems (CAS) theory, as given in the first section of this chapter.
- Practical Perspective dependent on
    - Years of experience with Deterministic software development methodologies.

- Consulting, practicing, and writing about Rapid Application Development (RAD) techniques; and working with high-innovation 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 depends on two of these ideas –

- Emergence
- Complexity

Emergence

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

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

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

- Each bird attempts to
  - Maintain a minimum distance from other objects in the environment, including different birds.
  - Match velocities with birds in its neighborhood.
  - Move towards the perceived center of mass of birds in its neighborhood.
- There are no principles of behavior for the group. The only principles are about the behavior of individual birds.
- However, there exists an emergent behavior, the flocking of birds. At the point When errant birds rush to catch up, the flock splits around obstacles and reforms on the different side.

This shows the prerequisite of the most troublesome mental model changes in Adaptive Development – From methods of managing and organizing that individual freedom to the idea that a innovative new order emerges unpredictably from unconstrained self organization.

In addition to the development, emergence is the most significant idea from the management point of view also.

Complexity

In the Software Development context, Complexity is about –

- The individuals of a team such as the developers, clients, sellers, competitors, and investors, their numbers and their speed.
- Size and technological complexity.

Adaptive Software Development Practices

Adaptive Software Development offers an alternate perspective on software management practices. In the sections below, you can understand the two significant practices – Quality and RAD, both of which have implications for gathering requirements.

You can discover the details of all the practices in the chapter, Adaptive Software Development Practices in this study notes.

Quality

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

RAD Practices

RAD Practices generally include a combination of the following –

- Evolutionary Lifecycle
- Client 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, rising flavor. Many IT organizations are against RAD. However, Microsoft and others have delivered incredibly large and complex software utilizing procedures comparable to RAD since it brings up questions about their fundamental world view.

RAD practices and Microsoft process are the two instances of Adaptive Development in action. Giving them a name (i.e., Adaptive Development) and understanding that there is a growing body of scientific knowledge (i.e., CAS theory) clarifies why they work. This should provide a basis for more extensive utilization of these practices.

Adaptive S/W Development - Lifecycle

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

Jim Highsmith published Adaptive Software Development in 2000.

Adaptive Software Development practices give ability to accommodate change and are adaptable in turbulent situations with products evolving with small planning and learning.

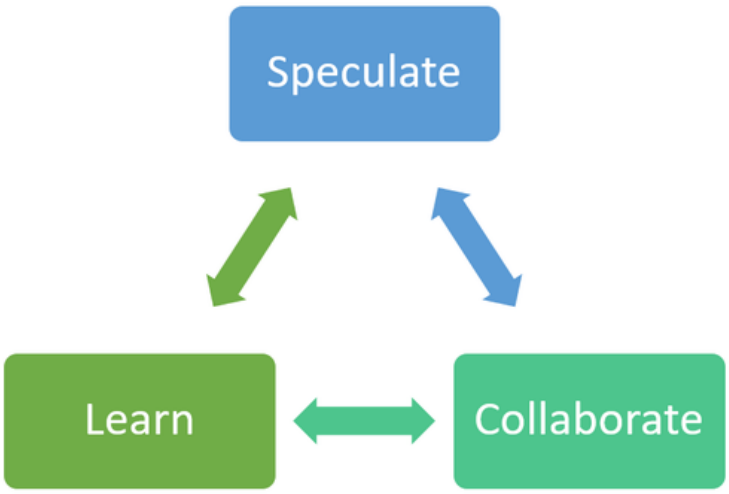
Stages of ASD Life Cycle

Adaptive Software Development is cyclical like the Evolutionary model, with the stage names reflecting the unpredictability in the complex systems. The stages in the Adaptive development life cycle are –

- Speculate
- Collaborate
- Learn

These three stages reflect the dynamic idea of Adaptive Software Development. The Adaptive Development explicitly replaces Determinism with Emergence. It goes beyond a negligible change in lifecycle to a more 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 outcomes are recognized as application highlights.



Speculate

The term plan is too deterministic and demonstrates a reasonably high degree of certainty about the desired outcome. The implicit and explicit objective of conformance to plan, restricts the supervisor's ability to steer the project in innovative ways.

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

Collaborate

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

Collaborate would require the capacity to work jointly to create results, share information or make decisions.

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

Learn

The Learn part of the Lifecycle is fundamental for the success of the project. Team has to improve their knowledge constantly, using practices such as −

- Technical Reviews
- Project Retrospectives
- Client Focus Groups

Reviews should be done after every emphasis. Both, the developers and clients examine their assumptions and utilize the results of every 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 cycles 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 see from the Speculate-Collaborate-Learn cycle, given above, it is obvious that the three stages are nonlinear and overlap.

We observe the following from an Adaptive framework.

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

Adaptive S/W Development - 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 comprehend these six characteristics of Adaptive Software Development.

Mission-focused

For many projects, the overall mission that guides the team is all around enunciated, however 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 goal. Mission statements and the conversations that result in those statements give guidance and criteria for making critical project tradeoff decisions.

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

Feature-based

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

Features can evolve more than a few iterations when the clients give feedback.

The application features that provide direct outcomes to the client after implementation are primary. A client-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 optional.

Iterative

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

Time-boxed

In Adaptive Software Development Lifecycle, the iterations are time-boxed. However, one should recall that time-boxing in Adaptive Software Development isn't about time deadlines. It should not

be utilized 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 Lifecycle, the emphasess are time-boxed. Nonetheless, one ought to recall that time-confining Adaptive Software Development isn't about time cutoff times. It ought not be utilized to make the cooperation for extended periods testing a communitarian situation or for settling on the nature of the expectations.

In Adaptive Software Development, time-boxing is considered as a direction for focusing and forcing hard tradeoff choices 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 timebox to get the work wrapped up.

Risk-driven

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

Change-tolerant

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

Adaptive S/W Development - Practices

The Adaptive Software Development practices are driven by a faith in continuous adaptation, with the lifecycle prepared to accepting continuous change as the standard.

Adaptive Software Development Lifecycle is dedicated to −

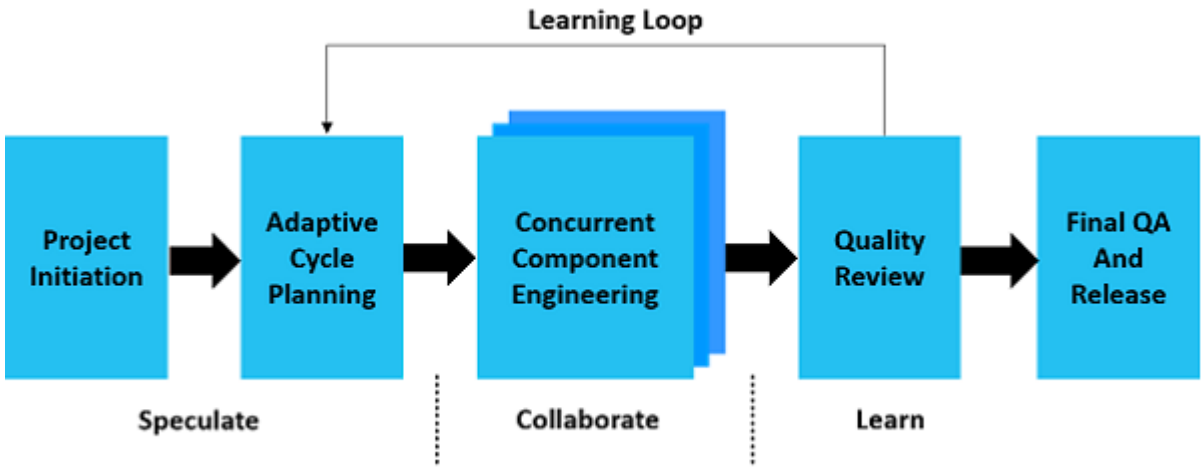
- Continuous learning
- Change direction
- Re-assessment
- Peering into an uncertain future
- Exceptional collaboration among developers, management, and clients

Adaptive SDLC

Adaptive Software Development consolidates RAD with Software Engineering Best Practices, for example −

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

Adaptive Software Development practices can be illustrated as following −



As mentioned above, Adaptive Software Development practices are spread across the three stages as following −

- **Speculate** − Initiation and planning
  - Project Initiation
  - Establishing time-box for the whole 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 client'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 commencement and planning stage. They are –

- Project initiation
- Establishing time-box for the whole 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 information should be gathered in a fundamental JAD session, considering speed as the major aspect. Initiation can be finished in a concentrated two to five day effort for a small to medium sized projects, or two to three weeks effort for bigger projects.

During the JAD sessions, prerequisites are gathered in enough detail to distinguish features and establish an overview of the object, information, or other architectural model.

Establishing Time-box for the Entire Project

The time-box for the whole project should be established, based on the scope, feature-set prerequisites, estimates, and asset availability that result from project commencement work.

As you aware, Speculating doesn't abandon estimating, however it just means accepting that estimates can go wrong.

Iterations and Time-box

Decide on the number of iterations and the individual iteration lengths dependent on the overall project scope and the degree of vulnerability.

For a small to medium sized application −

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

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

Develop a Theme or Objective

The team members should build up a theme or objective for each iteration. This is something like 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 client to enable review and feedback.

Within the iterations, the builds should deliver working highlights on a ideally 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 finish of the project.

Assign Features

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

During the task of features to the iterations −

- Development team should think of the feature estimates, risks, and dependencies and provide them to the client.

- Clients should decide on feature prioritization, using the information gave by the development team.

Thus iteration planning is feature-based and done as a team with developers and clients. Experience has indicated 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 stage, the focus is on the development. The Collaborate stage has two activities −

- The Development team collaborate and deliver working software.
- The project supevisors facilitate collaboration and concurrent development activities.

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

Teams should collaborate on −

- Technical problems
- Business requirements
- Quick decision making

Following are the practices relevant to the Collaborate stage in Adaptive Software Development −

- Collaboration for distributed teams
- Collaboration for smaller projects
- Collaboration for bigger 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 colleagues are working in physical proximity, Collaboration with informal hallway chats and whiteboard scribbling should be encouraged, as this is found to be powerful.

Collaboration for Bigger Projects

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

Learn - Quality Review

Adaptive Software Development empowers the idea of 'Experiment and Learn'.

Learning from the mistakes and experimentation requires that the colleagues 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 client'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 Client's Perspective

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

Client 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 emphasis.

Result Quality from a Technical Perspective

In the Adaptive Software Development projects, periodic survey of technical artifacts should be given importance. Code Reviews should be done on a continuous basis. Audits of other technical artifacts, such as technical architecture can be conducted weekly or at the finish 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 for example –

- Figure out 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, deciding the project status is feature-based approach, the finish of each iteration marked by finished 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 theoretical, more than the question 2 above, question 3 is significant. That is, the project team and the clients need to continuously ask themselves, "What have we learned until this point, and does it change our viewpoint on where we need to go?"

Adaptive S/W Development - Management

A flowchart of the Traditional software management is demonstrated below as follows.

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

Many organizations are steeped in a tradition of optimization, proficiency, consistency, control, rigor and process improvement. In any case, the developing 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 association, human-focused 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 need to consider the Adaptive management methods that have been demonstrated successful in different fields.

Adaptive Management

Adaptive management has demonstrated successful in the environments where the asset supervisors worked together with stakeholders and scientists as a team, with the following objectives −

- 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 asset management activities are experiments as their results cannot be reliably predicted already. These experiments are then used as learning opportunities for the improvements in the future.

Adaptive management is intended to build the capacity to respond timely in the face of new data 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 comprehended.

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

Adaptive management assists to change the decisions made by making it clear that −

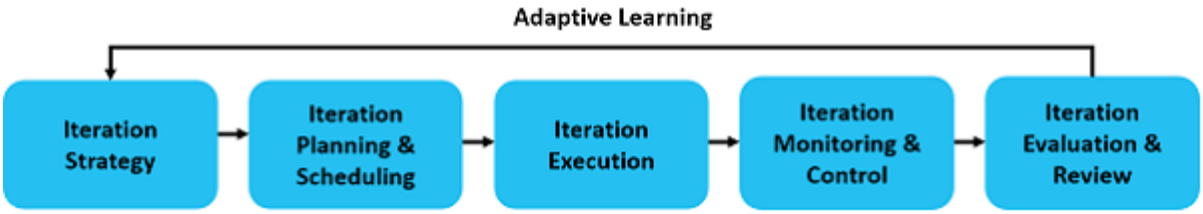
- The decisions are temporary.
- A management's decision need not always be right.
- Modifications are normal.

There are two types of Adaptive management approaches −

- Passive Adaptive Management.
- Active Adaptive Management.

Passive Adaptive Management

Adaptive management plans to improve the scientific knowledge and thereby reduce uncertainties.

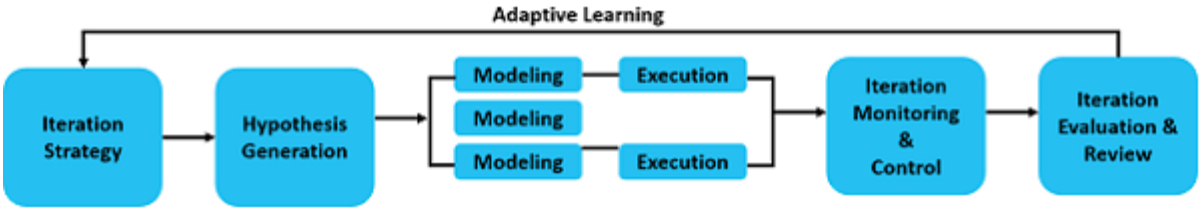


Within Passive Adaptive management, a single preferred course of action, based on existing data and understanding, is chosen. The results of management activities are observed, and subsequent decisions are adjusted based on the results.

This approach contributes to the learning and powerful management. In any case, it is restricted in its ability to enhance scientific and management abilities 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 utilizations), rather than a single model, is then developed. Management options are chosen dependent on the assessments of these alternative models.

Leadership-Collaboration Management

Adaptive management is what is most appropriate for Adaptive Software Development. The approach requires asset managers, for example 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 officers. 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 included and give guidance.
- Collaborate, facilitate and macro-manage the team.
- Give 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.

The end...