CHAPTER

7

Agile Model

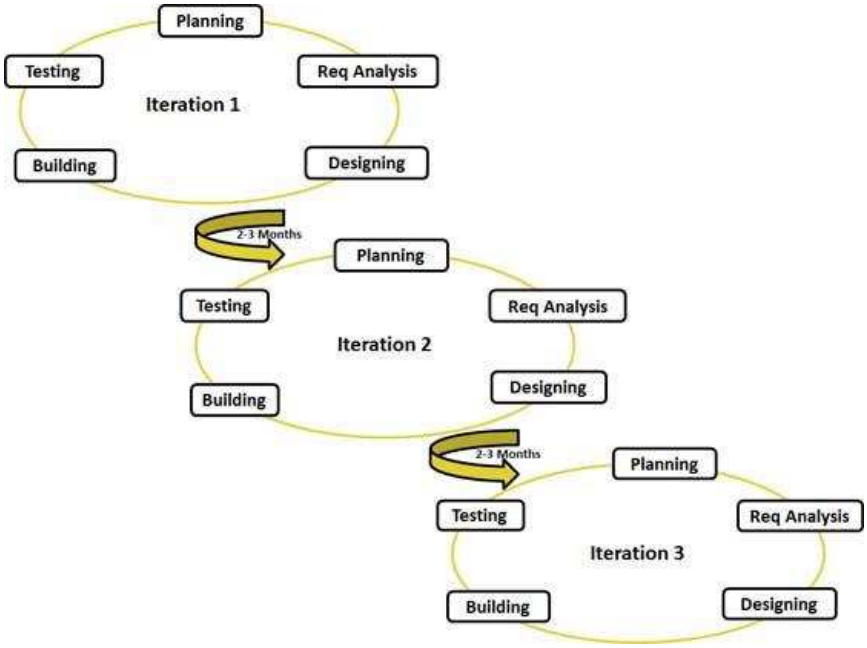
Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is displayed to the customer and important stakeholders.

What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release. Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model:



Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability. The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles:

- **Individuals and interactions** – in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software –** Demo working software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
- **Customer collaboration** – As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** – agile development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLC Models

Agile is based on the adaptive software development methods where as the traditional SDLC models like waterfall model is based on predictive approach.

Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle. Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model Pros and Cons

Agile methods are being widely accepted in the software world recently, however, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

Following table lists out the pros and cons of Agile Model

Pros	Cons
▪ Is a very realistic approach to software development ▪ Promotes teamwork and cross training. ▪ Functionality can be developed rapidly and demonstrated. ▪ Resource requirements are minimum. ▪ Suitable for fixed or changing requirements ▪ Delivers early partial working solutions. ▪ Good model for environments that change steadily. ▪ Minimal rules, documentation easily employed. ▪ Enables concurrent development and delivery within an overall planned context. ▪ Little or no planning required ▪ Easy to manage ▪ Gives flexibility to developers	▪ Not suitable for handling complex dependencies. ▪ More risk of sustainability, maintainability and extensibility. ▪ An overall plan, an agile leader and agile PM practice is a must without which it will not work. ▪ Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. ▪ Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction. ▪ There is very high individual dependency, since there is minimum documentation generated. ▪ Transfer of technology to new team members may be quite challenging due to lack of documentation.

CHAPTER

8

RAD Overview

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

What is RAD?

Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build, and test phases into a series of short, iterative development cycles. Following are the phases of RAD Model:

- **Business Modeling:**
  The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

- **Data Modeling:**

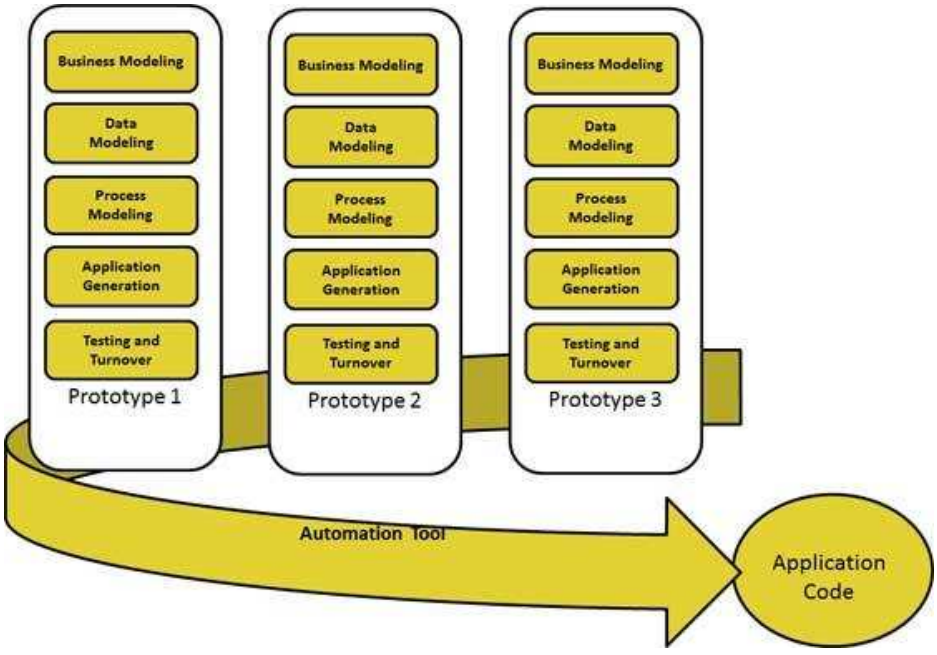
The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

- **Process Modeling:**
  The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding , deleting, retrieving or modifying a data object are given.

- **Application Generation:**
  The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

- **Testing and Turnover:**
  The overall testing time is reduced in RAD model as the prototypes are independently tested during every iteration. However the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

Following image illustrates the RAD Model:



RAD Model Vs Traditional SDLC

The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts a pressure on the customer to sign off the requirements before the project starts and the customer doesn't get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he actually gets to see the software, however the change process is quite rigid and it may not be feasible to incorporate major changes in the product in traditional SDLC.

RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non conformance with the actual user requirements.

RAD Model Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail. Following are the typical scenarios where RAD can be used:

- RAD should be used only when a system can be modularized to be delivered in incremental manner.
- It should be used if there's high availability of designers for modeling
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge
- Should be used where the requirements change during the course of the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD Model Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to reusability of the components and parallel development.

RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

Following table lists out the pros and cons of RAD Model

Pros	Cons
▪ Changing requirements can be accommodated. ▪ Progress can be measured. ▪ Iteration time can be short with use of powerful RAD tools. ▪ Productivity with fewer people in short time. ▪ Reduced development time. ▪ Increases reusability of components	▪ Dependency on technically strong team members for identifying business requirements. ▪ Only system that can be modularized can be built using RAD ▪ Requires highly skilled developers/designers. ▪ High dependency on modeling skills ▪ Inapplicable to cheaper projects as cost

▪ Quick initial reviews occur ▪ Encourages customer feedback ▪ Integration from very beginning solves a lot of integration issues.	of modeling and automated code generation is very high. ▪ Management complexity is more. ▪ Suitable for systems that are component based and scalable. ▪ Requires user involvement throughout the life cycle. ▪ Suitable for project requiring shorter development times.

CHAPTER

9

Software Prototyping Overview

The Software Prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

What is Software Prototyping?

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation. Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is the stepwise approach to design a software prototype:

- **Basic Requirement Identification**
  This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

- **Developing the initial Prototype**
  The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

- **Review of the Prototype**
  The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

- **Revise and enhance the Prototype**
  The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like , time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.

Prototypes can have horizontal or vertical dimensions. Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

- **Throwaway/Rapid Prototyping**
  Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

- **Evolutionary Prototyping**
  Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. Using evolutionary prototyping only well understood requirements are included in the prototype and the requirements are added as and when they are understood.

- **Incremental Prototyping**
  Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

- **Extreme Prototyping**
  Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.
Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as below.

Following table lists out the pros and cons of Big Bang Model

Pros	Cons
▪ Increased user involvement in the product even before implementation ▪ Since a working model of the system is displayed, the users get a better understanding of the system being developed. ▪ Reduces time and cost as the defects can be detected much earlier. ▪ Quicker user feedback is available leading to better solutions. ▪ Missing functionality can be identified easily ▪ Confusing or difficult functions can be identified	▪ Risk of insufficient requirement analysis owing to too much dependency on prototype ▪ Users may get confused in the prototypes and actual systems. ▪ Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans. ▪ Developers may try to reuse the existing prototypes to build the actual system, even when its not technically feasible ▪ The effort invested in building prototypes may be too much if not monitored properly

CHAPTER

10

SDLC Summary

This was about the various SDLC models available and the scenarios in which these SDLC models are used. The information in this tutorial will help the project managers decide what SDLC model would be suitable for their project and it would also help the developers and testers understand basics of the development model being used for their project.

We have discussed all the popular SDLC models in the industry, both traditional and Modern. This tutorial also gives you an insight into the pros and cons and the practical applications of the SDLC models discussed.

Waterfall and V model are traditional SDLC models and are of sequential type. Sequential means that the next phase can start only after the completion of first phase. Such models are suitable for projects with very clear product requirements and where the requirements will not change dynamically during the course of project completion.

Iterative and Spiral models are more accommodative in terms of change and are suitable for projects where the requirements are not so well defined, or the market requirements change quite frequently. Big Bang model is a random approach to Software development and is suitable for small or academic projects.

Agile is the most popular model used in the industry. Agile introduces the concept of fast delivery to customers using prototype approach. Agile divides the project into small iterations with specific deliverable features. Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment.

RAD (Rapid Application Development) and Software Prototype are modern techniques to understand the requirements in a better way early in the project cycle. These techniques work on the concept of providing a working model to the customer and stockholders to give the look and feel and collect the feedback. This feedback is used in an organized manner to improve the product.

The 'Resources' section below lists some suggested books and online resources to gain further understanding of the SDLC concepts.

Please send us your feedback at **webmaster@tutorialspoint.com**

Keep visiting to us, Happy Learning!

References

Books:

1) Lean Software Development: An Agile Toolkit for Software Development Managers - by Mary Poppendieck, Tom Poppendieck, Ken Schwaber

2) The Art Of Software Testing - By Glenford J Mayers

3) Agile Project Management with Scrum - by Ken Schwaber

4) Extreme Programming Explained - Book by Kent Beck

Websites:

http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

http://en.wikipedia.org/wiki/Agile_software_development

http://agilemanifesto.org/

http://www.agilealliance.org/