**Chapter 6: Agile Modelling and Prototyping**

Table of Contents

[Prototyping 4](#_Toc63530418)

[Prototyping as an Alternative to SDLC 5](#_Toc63530419)

[Advantages and Disadvantages of Prototyping 6](#_Toc63530420)

[Rapid Application Development 8](#_Toc63530421)

[Agile Modelling 10](#_Toc63530422)

[The Agile Manifesto Values 10](#_Toc63530423)

[The Agile Manifesto Principles 11](#_Toc63530424)

[Values and Motivation for Agile Modelling 12](#_Toc63530425)

[Activities of Agile Modelling 13](#_Toc63530426)

[Core Agile Practices 13](#_Toc63530427)

[Agile Development Process 14](#_Toc63530428)

[Scrum 16](#_Toc63530429)

[Scrum Features 17](#_Toc63530430)

[Scrum Team Roles 18](#_Toc63530431)

[Kanban 19](#_Toc63530432)

We have previously studied the System Development Life Cycle, which was a methodology to develop systems. In it, we saw that there were many different steps, such as user requirement analysis, feasibility analysis, etc.

SDLC has a few disadvantages though. We saw that it was difficult to make changes once we had completed one of the steps, which made it difficult to accommodate for changes in requirements from the users. Additionally, it is also very time consuming. This means that if, say, we make a system spending 20 months on it, by the time we release the system, a large portion of our work will be based on obsolete technology.

There is another development system called the Waterfall System, which allows for some changes. We have the ability to go back one step to accommodate some changes in that system. However, this is still not fully capable to getting rid of the disadvantages mentioned.

The Agile method of development solves these problems. It is incremental and iterative.

By incremental, we mean that a few features of the system are released every few months so that users can actually use those features. In this way, development takes less time and the overall system is not based on outdated technology but is always up-to-date.

By iterative, we mean that that the actual development happens in parts. Say the stages we have decided on are plan build test supply. These stages form a cycle, but we do not complete the entire system in one go. Each part of the system is built using the cycle and once a stage is complete, the cycle starts again for the next stage. In this way, if there are changes in requirements, we can easily repeat just the cycle for that stage instead of the whole thing.

## Prototyping

Closely related to the Agile development methodology is the idea of prototypes. Prototypes are like samples of the actual system. They can be of a few types:

* Patched-Up Prototypes – These are prototypes that are patched together, meaning the different features are not as well-made or efficient as they could be. For example, say we have a login page that works but it does not have a ‘Forgot Password’ option.
* Non-Operational Prototypes – These are used mostly with user interfaces. They provide a visual representation of what the system will look like but the interface is not actually usable.
* First-of-a-Series Prototypes – These are pilot prototypes. A single prototype is made of the entire system including all the features so that it can be tested and any problems that appear need to be fixed on just this one prototype. This may be because it would be risky or expensive to make a large number of prototypes and start testing them all. After everything is determined to be alright, more copies are made.
* Selected Features Prototypes – These are prototypes that include a few select features of the system, but the features work as intended.

Of course, the prototype can be a combination of different types as well.

### Prototyping as an Alternative to SDLC

The two major disadvantages we saw with SDLC were

* Going through the whole cycle took a long time
* If user requirements changed at any point, we had to start the cycle again

System development methodologies that work like this, where we complete the system step by step and make sure that everything on the current step is completed perfectly before moving on to the next one, fall under the category of Waterfall modelling. They are all very time consuming. We also have Iterative Modelling, where we create a part of the system and give it to the user to try out. Both Agile and Prototyping fall under this category.

Rather than using prototyping as a separate methodology, we can use it alongside SDLC. This will give us the advantages of both systems. We will overall still be working with the entire system step by step as we did in SDLC, but we will go over those steps repeatedly, completing a small part on each iteration and delivering it to the user.

The advantages of this are twofold.

* First, we can shorten the time between gathering requirements and delivering a working system. Since we are only developing a small part, we can get it into the hands of the user quickly.
* Second, we can better adapt to changes. Users will be able to actually see a part of the final product and will be able to give better feedback. We can use that feedback to improve our system as we are working on it, instead of making the entire thing, having the user tell us what things we did wrong and going back all over again.

There are also two concerns with doing this.

* Firstly, we are most likely trying to build something we do not completely understand yet, since we are only working with a part and not the whole system.
* Secondly, we might end up creating something that a few groups of users will be happy with, but which will not work properly in the complete system.

### Advantages and Disadvantages of Prototyping

Even if we do not use prototyping alongside SDLC, there are a few advantages and disadvantages to the prototyping methodology itself.

Disadvantages include:

* Creating prototypes for large systems can sometimes be difficult
* Users and even analysts might end up using the prototype as though it were a complete system, thus stopping development before it is actually complete.

Advantages, some of which has been indicated before, include

* The fact that we can make changes earlier on in development. We are able to deliver a working model to our users faster, take feedback and adapt to changes.
* We also have the opportunity to create something that closely addresses the users’ needs and expectations.
* If what we are doing is not working out, we can stop before we have wasted too much time and effort.

## Rapid Application Development

Rapid application development (RAD) is another development methodology. This is like the middle ground between the Waterfall model and the Iterating model. The goal is to speed up the process of development without making compromises in terms of quality.

In RAD, we have three phases:

1. Requirements Planning

Users and analysts meet to identify objectives of the application or system. Since the goal is to speed up the process of development, it is preferable to use JADs rather than say, interviews, which are time consuming.

1. RAD Design Workshop

This is subdivided into two parts.

First, we sit with the user and design a part of the system. That is one huge point in RAD, that the user is involved in every step. Once the system is designed, we move onto part two, which is to build the system. Building it should not take long, since again, we are trying to work fast. This should not be difficult since we are only working with a small part and not the whole system.

Once it is built, we take the built system back to the user and show it to them. They give feedback and we redesign and rebuild. Thus, the two steps go over and over just like it did with prototyping.

1. Implementation – This is just implementing everything we designed and built. We bring all the parts together and build the entire system.

Thus, the first and last stage in RAD are like the waterfall model, where we want to get things working perfectly before moving on. The stage in between is iterative, where we work piece by piece, show the user and adapt to feedback.

## Agile Modelling

Unlike the Waterfall model for system development, the Agile model has methods that are innovative, user-centred approaches to system development. The basic process is to define an overall system plan quickly and get into the development phase. After that, it is a repeated cycle of developing a part of the system, releasing it to users, collecting feedback and revising the system based on the feedback. This continuous revision with the help of user feedback is what eventually leads to a fully developed system.

We have previously discussed problems that we face using different system development methodologies. Because of these problems, The Agile Manifesto was created, which listed 4 values and 14 principles that should be followed in order to create productive software that users will be satisfied with.

### The Agile Manifesto Values

The 4 values are:

* **Individuals and interactions over processes and tools** – If we have problems with parts of the system, it is better to interact with users and other individuals involved in the development process and discuss the issue to come up with a solution rather than following a specific set of rules and steps that is meant to achieve the solution, as is done in the Waterfall model. Essentially, do not work like a robot.
* **Working software over comprehensive documentation** – In other development methodologies, there is a lot of documentation involved before actual development begins. Everything from user requirements to feasibility reports to layouts have to be documented before work can begin on the actual product. This takes up a huge amount of time. Agile recommends that we instead create a cycle of development, feedback collection and adjustment instead of spending time documenting everything.
* **Customer collaboration over contract negotiation** – Other development methodologies collect all the requirements from customers at the beginning and then build a system based on that. That is not how agile works. Here, we continuously work with customers to improve and change the system to better fit their needs. As such, we cannot sit around haggling over contracts and refuse to make even the smallest of changes simply because that is not what was agreed upon in the contract at the beginning. Instead, we should do our best to accommodate the requirements of our customers and make small changes if required in order to satisfy their needs.
* **Respond to change instead of following a plan** – This is similar to the previous points. We should not be headstrong and keep to the original plan, but instead try our best to adapt to changes and use other options if we come to realize that they would be better for the system. In Agile, we see changes as opportunities to improve the system rather than as obstacles.

### The Agile Manifesto Principles

The 14 principles of the agile manifesto are:

1. Satisfy the customer through delivery of working software.
2. Embrace change, even if introduced late in development. Accept changes as opportunities to improve the system.
3. Continue to deliver functioning software incrementally and frequently.
4. Encourage customers and analysts to work together daily.
5. Trust motivated individuals to get the job done.
6. Promote face-to-face conversation.
7. Concentrate on getting software to work.
8. Encourage continuous, regular, and sustainable development.
9. Adopt agility with attention to mindful design.
10. Support self-organizing teams.
11. Provide rapid feedback. Gather feedback from users quickly and work to improve the system immediately using the feedback.
12. Encourage quality.
13. Review and adjust behaviour occasionally.
14. Adopt simplicity. Use the simplest available solution for every part of the system.

### Values and Motivation for Agile Modelling

* **Communication** – Communication between developers and customers is more valuable than anything.
* **Simplicity** – The system should be kept as simple as possible. If something can be done in a simpler manner, we should do it.
* **Feedback** – We need to adapt to user feedback and improve our system.
* **Courage** – We need to have the courage to adapt to changes rather than seeing them as challenges to our plan.

### Activities of Agile Modelling

The activities involved in the Agile modelling process are:

* Listening to users
* Designing the system
* Coding
* Testing
* Repeat

At the end of each cycle, we make a release where the system is given to users and the process starts again.

### Core Agile Practices

There are four practices in Agile modelling that are essential to it:

* **Short Releases** – The cycles we mentioned should not take a long time. We should make changes quickly and make releases frequently so as to involve customers more with the development process.
* **40-Hour Work Weeks** – The development teams should work intensely during a typical 40-hour work week, but overtime is discouraged since it affects their health. The free time ensures they return to work relaxed and stress-free, which in turn improves work quality and prevents errors.
* **On-Site Customer** – A customer who actually understands the process of system development should be directly involved with the development process.
* **Pair Programming** – There should be at least 2 developers working on every part of the system. This will make the quality of the work better.

### Agile Development Process

1. Listen to user stories from the customer
2. Draw a logical workflow model to gain an appreciation for the business decisions represented in the user stories.
3. Create new user stories based on the logical model.
4. Develop some display prototypes.
5. Using feedback from the prototypes and the logical workflow diagrams, develop the system.

The only thing in the above steps that we are unfamiliar with at this point is User Stories, so we will be exploring that now.

#### User Stories

User stories are features and requirements for the system. These usually come from customers, but they can also come from administrators, managers or any other person using the system.

For example, these could be a few user stories:

* Display alternative flights
  + Prepare a list of the five cheapest flights
* Offer cheaper flights
  + Suggest that customers travel on other days, take special promotions or use alternate airports.
* Purchase a ticket
  + Allow customers to directly purchase tickets using credit cards (check validity).
* Allow customer to choose their seats
  + Direct the customers to a visual display of the seating arrangement and ask them to select a seat.

Notice that we gave a title and then a short description for each user story and how it will be implemented.

However, the above format is not commonly used. The common format goes somewhat like ‘As a \_\_\_\_\_ I want to \_\_\_\_\_ so I can \_\_\_\_\_’. For example:

* As a *Registered User* I want to *change my password* so I can *keep my account safe*.
* As an *Administrator* I want to *block users* so I can *avoid fraudulent posts*.

This follows the ‘Who, What and Why’ pattern. Note that the ‘Who’ part cannot refer to a specific person by name. It has to be a user group.

User stories are divided into three categories:

* To Be Done
* Doing
* Done

## Scrum

Scrum is a type of Agile development methodology. In scrum, the process goes:

1. **Plan** – We have to figure out what we want to work on and how we are going to do it.
2. **Build** – Next, we build the features we had planned.
3. **Test** – We test the features we built.
4. **Review** – We review the process we went through to build the selected features. Perhaps we spent too much time planning and not enough testing. We need to improve upon this in the next cycle.

After each cycle we have a release, which is a working model. Using the takeaways from the review stage, we repeat the cycle for other parts of the system. We keep doing this again and again until we have a final release. Note that each release is an incremental release, meaning we are adding features to an existing system in each cycle.

The processes we go through under the scrum development model are combinedly called a sprint. A single sprint lasts about three to four weeks. We can have multiple sprints in a scrum.

### Scrum Features

A scrum consists of a few features:

* **Product Backlog** – This is the list of features that we need to develop. It is essentially a list of the user stories we collected. The list can change over time as requirements are adapted to feedback from users.
* **Sprint Backlog** – This is the list of features that are to be developed in the current sprint. The features are taken from the product backlog. Features with higher priorities are taken first.
* **Sprint**
* **Daily Scrum** – These are stand-up meetings where the team of developers, managers and anyone else involved meets. They happen every day and issues like who has made how much progress with the work at hand and any problems being faced are discussed. The reason they are called stand-up meetings is because everyone is literally standing since the meeting is 15 minutes at most.
* **Demo** – These are the releases that are made at the end of each sprint.

### Scrum Team Roles

[Reference](https://www.visual-paradigm.com/scrum/what-are-the-three-scrum-roles)

Every member participating in a scrum falls into one of three roles:

* **Product Owner** – The product owner is responsible for identifying features, prioritizing them and keeping track of profits and losses. They are like a representative for the customers.
* **Development Team** – This group consist of anyone and everyone related to the actual development, including programmers, UI designers, system analysts, etc.
* **Scrum Master** – The scrum master helps the team achieve a successful scrum. They are not like the manager, since they do not really manage the team. Instead, they help the team adopt Agile development practices, help remove problems and protect the team from outside interference.

Together, these members are called the Scrum Team (\*superhero background music\*).

## Kanban

[Reference](https://www.youtube.com/watch?v=rIaz-l1Kf8w)

Kanban and Scrum are similar in that they both try to ensure that features from the product backlog get to the customer as a demo very quickly. However, they handle the process a little differently.

Unlike Scrum, there are just three parts to the Kanban process:

* **Build**
* **Test**
* **Review**

The actual work in these processes is the same as their Scrum counter parts. The difference is that they handle features not in groups like a Sprint, but individually. Depending on the size of the development team, the build stage has a few features to work with, typically a small list. There is no Sprint Backlog mind. When something is built, it goes to the testing stage and from there to the review stage. As things leave the build stage, the development team decides which high priority features need to be taken in next and start work accordingly.

Other than this, the rest of the process is pretty similar. There are still daily stand-up meetings, demos, etc.