**Week 2: SDLC and process models**

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software’s. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

**SDLC Models**

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models.

Following are the most important and popular SDLC models followed in the industry −

* Waterfall Model
* Iterative Model
* Spiral Model
* V-Model
* Big Bang Model
* Agile model

**How to choose process model?**

## Factors in choosing a software process

Choosing the right software process model for your project can be difficult. If you know your requirements well, it will be easier to select a model that best matches your needs. You need to keep the following factors in mind when selecting your software process model:

### Project requirements

Before you choose a model, take some time to go through the project requirements and clarify them alongside your organization’s or team’s expectations. Will the user need to specify requirements in detail after each iterative session? Will the requirements change during the development process?

### Project size

Consider the size of the project you will be working on. Larger projects mean bigger teams, so you’ll need more extensive and elaborate project management plans.

### Project complexity

Complex projects may not have clear requirements. The requirements may change often, and the cost of delay is high. Ask yourself if the project requires constant monitoring or feedback from the client.

### Cost of delay

Is the project highly time-bound with a huge cost of delay, or are the timelines flexible?

### Customer involvement

Do you need to consult the customers during the process? Does the user need to participate in all phases?

### Familiarity with technology

This involves the developers’ knowledge and experience with the project domain, software tools, language, and methods needed for development.

### Project resources

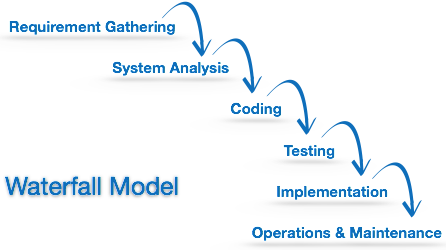
This involves the amount and availability of funds, staff, and other resources.

**Comparison between a defined process and an empirical process:**

|  |  |
| --- | --- |
| **Defined Process** | **Empirical Process** |
| The process is planned in detail before the project begins. | The planning takes place as the team progresses with the project. |
| The entire project is completed within a single cycle which may take months. | The entire project is completed in short cycles where it is reviewed after each cycle. |
| The project manager knows most of the information of the project. | Everyone involved in the project has an equal right to know about the details of the project. |
| There is a command and control type of leadership among the manager and the developers. | The Scrum or Agile team is self-organizing and have the liberty to ask questions and share their opinions. |
| The accountability of the project is on the project manager. | The accountability of the project is on everyone who is creating and delivering the product. |
| The time to market and return on investment takes longer. | The time to market and return on investment takes a shorter time. |
| The developers are told how and what they have to create and there is no scope of creativity or freedom. | The developers are given an idea such as a user story where they could employ any techniques or software and deliver the result according to their judgement. |

**Traditional process models:**

### Waterfall model

Waterfall model is the simplest model of software development paradigm. It says the all the phases of SDLC will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on.

This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. This model does not work smoothly if there are some issues left at the previous step. The sequential nature of model does not allow us go back and undo or redo our actions.

This model is best suited when developers already have designed and developed similar software in the past and is aware of all its domains.

**Advantages:**  
These are some advantages of Waterfall Model.

1. It is simple and easy to understand and use.
2. It is easy to manage.
3. It works well for smaller and low budget projects where requirements are very well understood.
4. Clearly defined stages and well understood.
5. It is easy to arrange tasks.
6. Process and results are well documented.

**Disadvantages:**  
These are some disadvantages of Waterfall Model.

1. It is difficult to measure progress within stages.
2. Poor model for long and ongoing projects.
3. No working software is produced until late during the life cycle.
4. High amounts of risk and uncertainty.
5. Not a good model for long and object oriented projects.
6. Cannot accommodate changing requirements.

### Incremental model

Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.



Each iteration passes through the **requirements, design, coding and testing phases**. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.

**Requirement Analysis:** Requirement and specification of the software are collected.

**Design:** Some high-end function are designed during this stage

**Code:** Coding of software is done during this stage

**Test:** Once the system is deployed, it goes through the testing phase

**Advantages of incremental Model**

* The software will be generated quickly during the software life cycle
* It is flexible and less expensive to change requirements and scope
* Throughout the development stages changes can be done
* This model is less costly compared to others
* A customer can respond to each building
* Errors are easy to be identified.

**Disadvantages of incremental Model**

* It requires a good planning designing
* Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle.
* Each iteration phase is rigid and does not overlap each other
* Rectifying a problem in one unit requires correction in all the units and consumes a lot of time

### Agile manifesto:

Agile is a time-bound, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver all at once.

The Agile approach to software development is defined by its commitment to creating software incrementally -- or in regular increases and stages. The approach offers users new versions, or releases, of software following brief periods of work. Those brief periods of work are often called [sprints](https://www.techtarget.com/searchsoftwarequality/definition/Scrum-sprint).

**The four core values of** [**Agile software development**](https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development) **as stated by the Agile Manifesto are:**

* individuals and interactions over processes and tools;
* working software over comprehensive documentation;
* customer collaboration over contract negotiation; and
* responding to change over following a plan.

**Twelve Principles of Agile Manifesto**

* **Customer Satisfaction** − Highest priority is given to satisfy the requirements of customers through early and continuous delivery of valuable software.
* **Welcome Change** − Changes are inevitable during software development. Ever-changing requirements should be welcome, even late in the development phase. Agile processes should work to increase customers' competitive advantage.
* **Deliver a Working Software** − Deliver a working software frequently, ranging from a few weeks to a few months, considering shorter time-scale.
* **Collaboration** − Business people and developers must work together during the entire life of a project.
* **Motivation** − Projects should be built around motivated individuals. Provide an environment to support individual team members and trust them so as to make them feel responsible to get the job done.
* **Face-to-face Conversation** − Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team.
* **Measure the Progress as per the Working Software** − Working software is the key and it should be the primary measure of progress.
* **Maintain Constant Pace** − Agile processes aim towards sustainable development. The business, the developers, and the users should be able to maintain a constant pace with the project.
* **Monitoring** − Pay regular attention to technical excellence and good design to enhance agility.
* **Simplicity** − Keep things simple and use simple terms to measure the work that is not completed.
* **Self-organized Teams** − An agile team should be self-organized and should not depend heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams.
* **Review the Work Regularly** − Review the work done at regular intervals so that the team can reflect on how to become more effective and adjust its behavior accordingly.

**Best Practices for Agile Teams**

### Collaborate with the customer

The customer is satisfied when requirements are fulfilled, expectations are met, and wants and needs are gratified.

### Work together daily

Teamwork, contribute to successful projects when they collaborate as a cohesive unit.

* Communication
* Coordination
* Balance of team member contributions
* Mutual support
* Effort
* Cohesion

### Build projects around motivated individuals

It takes motivation to push through an intense development cycle and get the work done right. Agile teams are passionate about their work, focused on the team goal, and supportive of each other. When there is trust and respect among peers, Agile teams establish a rhythm to their work that is fast-paced and predictable.

### Convey information face-to-face

Whether working through a knotty problem with a teammate or reporting on the day’s accomplishments at a daily meeting, Agile team members prefer face-to-face communication. Information lost in a full email box or voice message queue slows or blocks progress. The daily meeting is one time the entire team connects to find out if there are any issues that could cause delays. Face-to-face conversation opens channels and builds trust, making the Agile method sustainable.

### Form self-organizing teams

Choose how they will execute the work, and who will do what. They divide the work into increments that can be completed within each iteration, and into tasks that can be completed each day. Management does not assign tasks or look over their shoulders. The team is entrusted with making the right decisions. For this arrangement to work, each team member has to be confident in their work and commit to pushing through the most difficult, frustrating blocks.

### Reflect on how teams can become more effective

Agile teams routinely examine their performance and look for ways to do better. In fact, they are dedicated to continuous improvement. As individuals and as a group, teams are open to mentoring and coaching but, as one Agile coach notes, they don’t respond well to command and control behavior from management.

### 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.



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.

**The advantages of the Agile Model are as follows**

* 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.

**The disadvantages of the Agile Model are as follows**

* 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 a 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.

**Three Paradigm Shifts of Agile**

1. Triple-Constraints of Project Management (aka – The Iron Triangle)
2. Traditional vs Emergent Requirements
3. Project Managers vs ScrumMasters

**Triple-Constraints of Project Management (aka – The Iron Triangle)**

An Agile approach to a project is much more thoughtful about the cadence of the project. How long are our iterations (we fix that), who is on the dedicated cross-functional team (fixed run-rate budget), and what is the vision of what we are trying to achieve. We then pick a release planning approach and move forward keeping the team and the sprint as sacred as possible. That leaves us the ability to focus our cognitive processing ability not on managing change requests, re-baselining project plans, or updating requirements documents – but instead we are focusing on delivering the right thing.

**Traditional vs Emergent Requirements**

In traditional projects all the requirements of a project are specified upfront at a very precise level of detail. And because our projects are set up for failure having fixed the scope, cost, and time – when we start coming close to the end of the timeline or the end of the budget we inevitably have to de-scope something so we leave inventory of the table and deliver less that what we originally specified upfront. This is a significant waste within software development and is a key driver of why there are trust issues between business and IT groups.

Instead of specifying everything at great detail upfront and delivering something less, an Agile framework turns that upside down. We want to focus first on a very well defined Vision, decompose that into Themes (large chunks of functionality), and based on the priority of those Themes start elaborating more specific detail of the features, and then the highest priority features into stories, and the highest priority stories into technical designs and ultimately working software.

**Project Managers vs ScrumMasters**

Agile, on the other hand doesn’t require a Project Management role. Instead we look to the role of ScrumMaster. And ScrumMasters work for teams. Teams that are self-organizing and cross-functional. Teams that are in control of their own destiny and ultimately their own ability to succeed or fail. Instead of command and control, ScrumMasters act as Servant Leaders that insulate the team from distractions, remove impediments, and provide the tools and resources needed so the team can maximize success.