SOFTWARE ENGINEERING FUNDEMENTALS

Software Evolution

In software engineering, software evolution is referred to as the process of developing, maintaining, and updating software for various reasons. Software changes are inevitable because there are many factors that change during the life cycle of a piece of software.

Some of these factors include:

- Requirement changes
- Environment changes
- Errors or security breaches
- New equipment added or removed, and finally
- Improvements to the system

For many companies, one of their largest investments in their business is for software and software development. Software is considered a very critical asset, and management wants to ensure they employ a team of software engineers who are devoted to ensuring that the software system stays up to date with ever evolving changes.

Software Maintenance:

With software ever changing, it's important for software engineers to ensure they perform periodic maintenance on the software. There are four different types of software maintenance: corrective, adaptive, perfective, and preventive.

Corrective maintenance is the most common type of maintenance that addresses bugs, errors, faults in software that has already been released and is being used by the users. Sometimes, some of the defects are reported by the users themselves.

Adaptive maintenance deals with an environment change for the software. These changes could include hardware, software, operating system, and organizational policy changes. Any changes in the software's environment will usually cause for changes to be made in the software to ensure that the software will work in the new environment without any hiccups.

Perfective maintenance is concerned with making functional changes that are new or based on user requirements. This includes adding new functionalities or enhancements to the existing software that can help to increase the systems performance. And finally, **preventive maintenance** is concerned with making changes that ensures the longevity of the software. This may include code optimizing, code restructuring, and documentation.

Software Evolution Laws:

Lehman has given laws for software evolution. He divided the software into three different categories:

- **S-type (static-type) -** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.

- **P-type (practical-type) -** This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.

- **E-type (embedded-type) -** This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real-world situations. For example, Online trading software.

E-Type software evolution

Lehman has given eight laws for E-Type software evolution -

- **Continuing change -** An E-type software system must continue to adapt to the real-world changes, else it becomes progressively less useful.
- **Increasing complexity -** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.
- **Conservation of familiarity -** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.
- **Continuing growth-** For an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
- **Reducing quality -** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
- **Feedback systems-** The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation -** E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability -** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:

Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Models

Software development life cycle (SDLC) models show the ways to navigate through the complex and demanding process of software building. A project's quality, timeframes, budget, and ability to meet the stakeholders' expectations largely depend on the chosen model.

Today, there are more than 50 recognized SDLC models in use. The most used, popular and important SDLC models are given below:

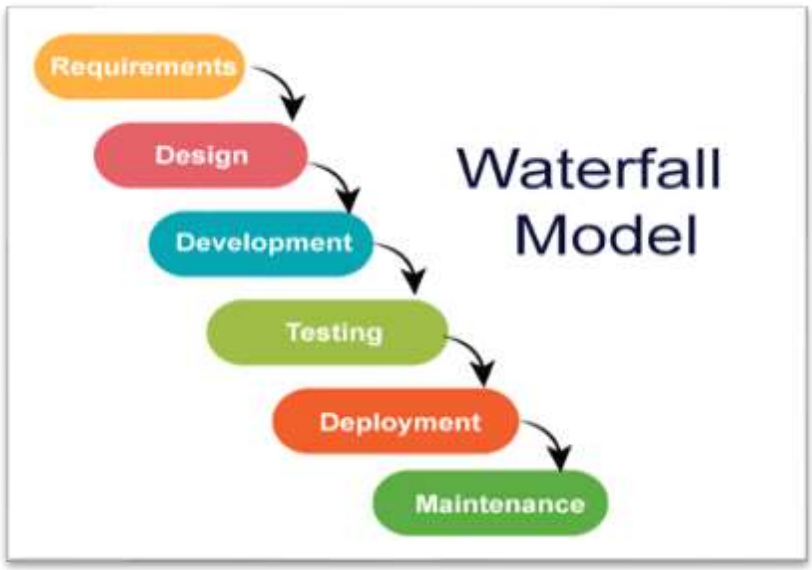
1. Waterfall model
2. V-model
3. Iterative model
4. Spiral model
5. Agile model

SDLC MODELS

1. Waterfall SDLC Model

The Waterfall model was the first approach to software development. As the name indicates, the process involves moving down through the linear development stages in order: analysis, design, development, testing, deployment, and maintenance. Every stage is well defined with specific deliverables and milestones.

The Waterfall model is sequentially linear, meaning the next step cannot begin until the current stage is complete. A stage is complete when the goals are met and someone signs off for the project to advance.

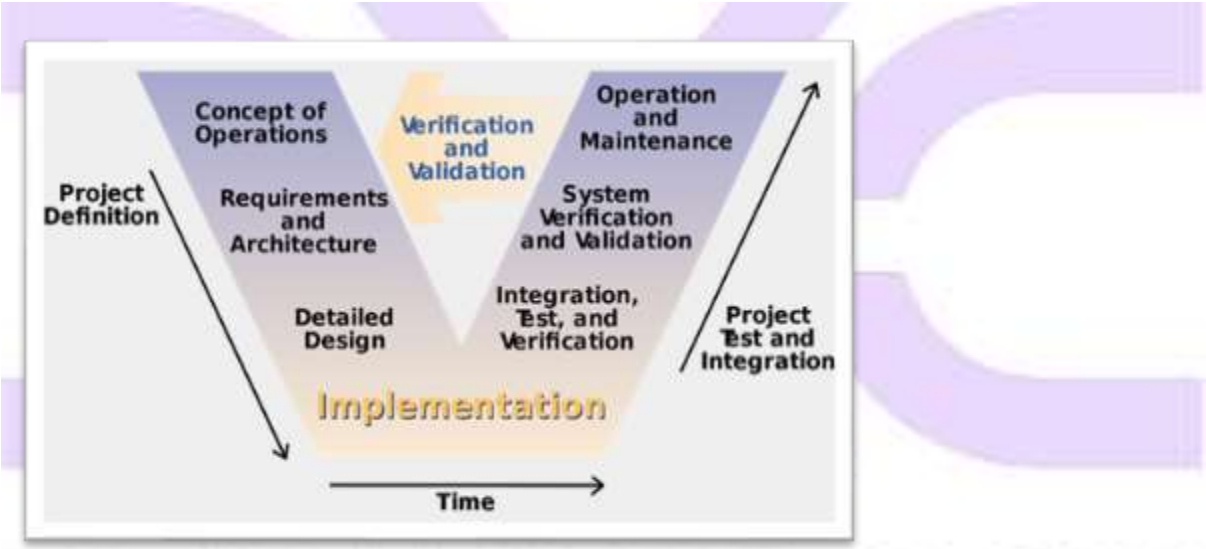


Advantages: Simple and understandable, the Waterfall Model is a manageable method ideal for lifecycle management of smaller projects where the requirements are established and finalized upfront.

Disadvantages: Because of its rigid structure, the Waterfall Model does not work well for complex projects where there is a chance of a change in requirements and/or significant impromptu testing throughout the software development stage.

2. V-model (Validation and Verification model)

The V-model is another linear model with each stage having a corresponding testing activity. Such workflow organization implies exceptional quality control, but at the same time, it makes the V-model one of the most expensive and time-consuming models. Moreover, even though mistakes in requirements specifications, code and architecture errors can be detected early, changes during development are still expensive and difficult to implement. As in the Waterfall case, all requirements are gathered at the start and cannot be changed.

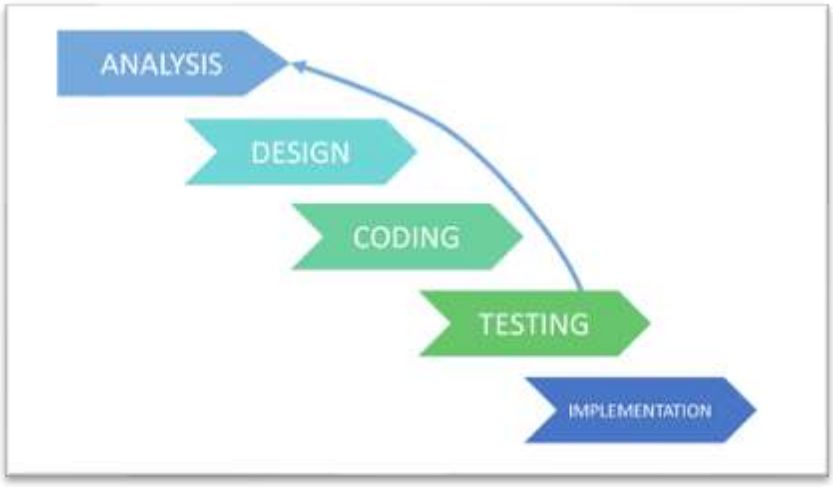


Advantages: The V-Model is a simple process that's great for smaller projects. Using the V-Model can yield a higher chance of success due to the test plans of the development stage and regularly schedule updates throughout its lifecycle.

Disadvantages: Similar to the Waterfall Model, the V-Model is very rigid in nature so it isn't ideal for applications or systems software that may require unforeseen changes/updates throughout the software lifecycle.

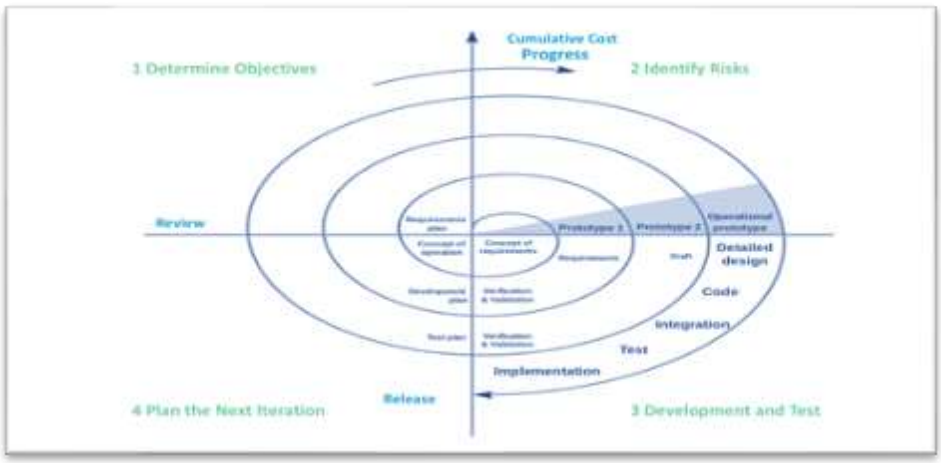
3. Iterative model

The Iterative SDLC model does not need the full list of requirements before the project starts. The development process may start with the requirements to the functional part, which can be expanded later. The process is repetitive, allowing to make new versions of the product for every cycle. Every iteration (which last from two to six weeks) includes the development of a separate component of the system, and after that, this component is added to the functional developed earlier. Speaking with math terminology, the iterative model is a realization of the sequential approximation method; that means a gradual closeness to the planned final product shape.



Advantages: The Iterative Model is a great solution for projects that need accommodation for some change requests between increments. This model also yields the benefit of being able to detect problems earlier in the software development for better lifecycle management planning.
Disadvantages: A potential disadvantage to the Iterative Model is the need for strategic planning and documentation. This method also tends to require more resources, staff and monetary, behind the project. This model isn't ideal for ongoing development as the next sequence cannot begin until the previous stage has fully completed.

4. **Spiral model**
The Spiral model puts focus on thorough risk assessment. Thus, to reap the benefits of the model to the fullest, you'll need to engage people with a strong background in risk evaluation. A typical Spiral iteration lasts around 6 months and starts with 4 important activities - thorough planning, risk analysis, prototypes creation, and evaluation of the previously delivered part. Repeated spiral cycles seriously extend project timeframes.
This is the model where intensive customer involvement appears. They can be involved in the exploration and review stages of each cycle. At the development stage, the customer's amendments are not acceptable.

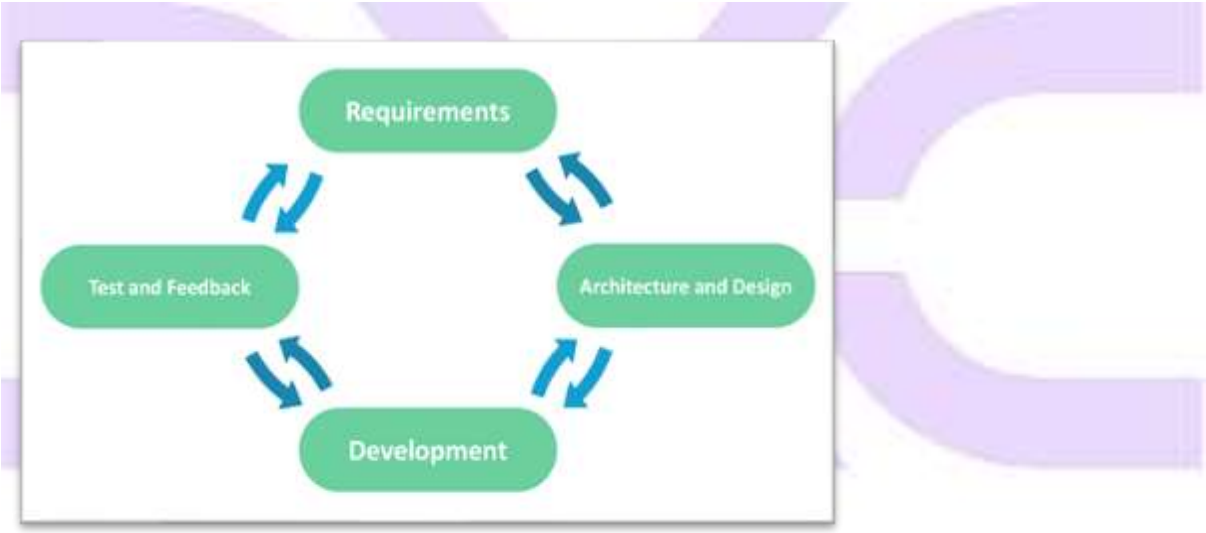


Advantages: The Spiral Model can be advantageous as it manages risks and divides development into phases. It also helps with more accurate estimates for budget and schedule as roadblocks are discovered earlier.

Disadvantages: Since this model is highly customized, repurposing the process can be confusing. It also requires team members that are well-versed in risk evaluation.

5. Agile model

In the agile methodology after every development iteration, the customer is able to see the result and understand if he is satisfied with it or he is not. This is one of the advantages of the agile software development life cycle model. One of its disadvantages is that with the absence of defined requirements it is difficult to estimate the resources and development cost. Extreme programming is one of the practical use of the agile model. The basis of such model consists of short weekly meetings – Sprints which are the part of the Scrum approach.



Advantages: The Agile Model decreases the amount of time to yield individual system features. It also calls for a lot of communication and continuous feedback from the customer/user that can provide clear direction for the project.

Disadvantages: The Agile method can potentially veer-off track as it relies on end-user interaction that may or may not be clearly expressed. Documentation is also minimal for an Agile software development strategy and requires a well-versed, cross-functional team.

<u>Software Development Paradigm</u>

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various research and requirement gathering which helps the software product to build. It consists of -

1. Requirement gathering
2. Software design
3. Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes -

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes -

- Coding
- Testing
- Integration

SDLC - Software Development Life Cycle Overview

SDLC or the Software Development Life Cycle refers to a methodology with clearly defined process that produces software with the highest quality and lowest cost in the shortest time possible. SDLC provides a well-structured flow of phases that help an organization to quickly produce high-quality software which is well-tested and ready for production use.

In detail, the SDLC methodology focuses on the following phases of software development:

No matter what type of the models has been chosen, each of them has basic stages which are used by every software development company. Let's explore those stages as this is important for the understanding of the each of SDLC models and the differences between them.

1. Planning and requirement analysis

Each software development life cycle model starts with the analysis, in which the stakeholders of the process discuss the requirements for the final product. The goal of this stage is the detailed definition of the system requirements. Besides, it is needed to make sure that all the process participants have clearly understood the tasks and how every requirement is going to be implemented. Often, the discussion involves the QA specialists who can interfere the process with additions even during the development stage if it is necessary.

2. Designing project architecture

At the second phase of the software development life cycle, the developers are actually designing the architecture. All the different technical questions that may appear on this stage are discussed by all the stakeholders, including the customer. Also, here are defined the technologies used in the project, team load, limitations, time frames, and budget. The most appropriate project decisions are made according to the defined requirements.

3. Development and programming

After the requirements approved, the process goes to the next stage – actual development. Programmers start here with the source code writing while keeping in mind previously defined requirements. The system administrators adjust the software environment, front-end programmers develop the user interface of the program and the logics for its interaction with the server.
The programming by itself assumes four stages

- ✓ Algorithm development
- ✓ Source code writing
- ✓ Compilation
- ✓ Testing and debugging

4. Testing

The testing phase includes the debugging process. All the code flaws missed during the development are detected here, documented, and passed back to the developers to fix. The testing process repeats until all the critical issues are removed and software workflow is stable.

5. Deployment

When the program is finalized and has no critical issues – it is time to launch it for the end users. After the new program version release, the tech support team joins. This department provides user feedback; consult and support users during the time of exploitation. Moreover, the update of selected components is included in this phase, to make sure, that the software is up-to-date and is invulnerable to a security breach.

How the SDLC Works

SDLC works by lowering the cost of software development while simultaneously improving quality and shortening production time. SDLC achieves these apparently divergent goals by following a plan that removes the typical pitfalls of software development projects. That plan starts by evaluating existing systems for deficiencies.

Next, it defines the requirements of the new system. It then creates the software through the stages of analysis, planning, design, development, testing, and deployment. By anticipating costly mistakes like failing to ask the end-user or client for feedback, SLDC can eliminate redundant rework and after-the-fact fixes.
It's also important to know that there is a strong focus on the testing phase. As the SDLC is a repetitive methodology, you must ensure code quality at every cycle. Many organizations tend to spend few efforts on testing while a stronger focus on testing can save them a lot of rework, time, and money. Be smart and write the right types of tests.

<u>Agile Concept</u>

Agile is a term used to describe software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a

disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

The four values of Agile

In 2001, 17 software development professionals gathered to discuss concepts around the idea of lightweight software development and ended up creating the Agile Manifesto. The Manifesto outlines the four core values of Agile, and although there has been debate about whether the Manifesto has outlived its usefulness, it continues at the core of the Agile movement.

The four core values outlined in the Agile Manifesto are:

Individual interactions are more important than processes and tools. People drive the development process and respond to business needs. They are the most important part of development and should be valued above processes and tools. If the processes or tools drive development, then the team will be less likely to respond and adapt to change and, therefore, less likely to meet customer needs.

A focus on working software rather than thorough documentation. Before Agile, a large amount of time was spent on documenting the product throughout development for delivery. The list of documented requirements was lengthy and would cause long delays in the development process. While Agile does not eliminate the use of documentation, it streamlines it in a way that provides the developer with only the information that is needed to do the work -- such as user stories. The Agile Manifesto continues to place value on the process of documentation, but it places higher value on working software.

Collaboration instead of contract negotiations. Agile focuses on collaboration between the customer and project manager, rather than negotiations between the two, to work out the details of delivery. Collaborating with the customer means that they are included throughout the entire development process, not just at the beginning and end, thus making it easier for teams to meet the needs of their customers. For example, in Agile software development, the customer may be included at different intervals for demos of the product. However, the customer could also be present and interacting with the teams on a daily basis, attending all meetings and ensuring the product meets their desires.

A focus on responding to change. Traditional software development used to avoid change because it was considered an undesired expense. Agile eliminates this idea. The short iterations in the Agile cycle allow changes to easily be made, helping the team modify the process to best fit their needs rather than the other way around. Overall, Agile software development believes change is always a way to improve the project and provide additional value.

The 12 principles of Agile

The Agile Manifesto also outlined 12 core principles for the development process. They are:

1. Satisfy customers through early and continuous delivery of valuable work.
2. Break big work down into smaller tasks that can be completed quickly.

3. Recognize that the best work emerges from self-organized teams.
4. Provide motivated individuals with the environment and support they need and trust them to get the job done.
5. Create processes that promote sustainable efforts.
6. Maintain a constant pace for completed work.
7. Welcome changing requirements, even late in a project.
8. Assemble the project team and business owners on a daily basis throughout the project.
9. Have the team reflect at regular intervals on how to become more effective, then tune and adjust behaviour accordingly.
10. Measure progress by the amount of completed work.
11. Continually seek excellence.
12. Harness change for a competitive advantage.

Types of Agile methodologies

The goal of every Agile methodology is to embrace and adapt to change while delivering working software as efficiently as possible. However, each method varies in the way it defines the steps of software development. The most widely used Agile methods include:

- Scrum
- Lean software development
- Extreme programming
- Crystal
- Kanban
- Dynamic systems development method
- Feature-driven development

Scrum is a lightweight Agile framework that can be used by project managers to control all types of iterative and incremental projects. In Scrum, the product owner creates a product backlog that allows them to work with their team to identify and prioritize system functionality. The product backlog is a list of everything that needs to be accomplished to deliver a successful, working software system -- this includes bug fixes, features and non-functional requirements. Once the product backlog is defined, no additional functionality can be added except by the corresponding team.

Once the team and the product owner have established the priorities, cross-functional teams step in and agree to deliver working increments of software during each sprint -- often within 30 days. After each sprint, the product backlog is revaluated, analysed and reprioritized in order to select a new set of deliverable functions for the next sprint. Scrum has gained popularity over the years since it is simple, has proven to be productive and can incorporate the various overarching practices promoted by the other Agile methods.

Lean software development is another iterative method that places a focus on using effective value stream mapping to ensure the team delivers value to the customer. It is flexible and evolving; it does not have rigid guidelines or rules. The Lean method uses the following primary principles:

- Increasing learning
- Empowering the team
- Fostering integrity
- Removing waste
- Understanding the whole
- Making decisions as late as possible
- Delivering the product as fast as possible

The Lean method relies on fast and reliable feedback between the customers and programmers in order to provide fast and efficient development workflows. To accomplish this, it provides individuals and small teams with decision-making authority instead of relying on a hierarchical flow of control. To eliminate waste, the Lean method asks users to only select truly valuable features for their system, prioritize these chosen features and then deliver them in small batches. Lean software development also encourages automated unit tests to be written simultaneously with the code and concentrates on ensuring every member of the team is as productive as possible.

The **Extreme programming (XP)** method is a disciplined approach that focuses on speed and continuous delivery. It promotes increased customer involvement, fast feedback loops, continuous planning and testing and close teamwork. Software is delivered at frequent intervals -- usually every one to three weeks. The goal is to improve software quality and responsiveness when faced with changing customer requirements.

The XP method is based on the values of communication, feedback, simplicity and courage. Customers work closely with their development team to define and prioritize their requested user stories. However, it is up to the team to deliver the highest priority user stories in the form of working software that has been tested at each iteration. To maximize productivity, the XP method provides users with a supportive, lightweight framework that guides them and helps ensure the release of high-quality enterprise software.

Crystal is the most lightweight and adaptable methodology. It focuses on people and the interactions that occur while working on an Agile project as well as business-criticality and priority of the system under development. The Crystal method works off of the realization that every project possesses unique characteristics that require a slightly tailored set of policies, practices and processes. As a result, it is made up of a collection of Agile process models, such as Crystal Orange, Crystal Clear and Crystal Yellow. Each model has its own unique characteristics that are driven by different factors, including project priorities, team size and system criticality.

Similar to other Agile methodologies, Crystal emphasizes frequent delivery of working software with high customer involvement, adaptability and the elimination of bureaucracy and distractions. Its key principles include communication, teamwork and simplicity.

Kanban uses a highly visual workflow management method that allows teams to actively manage product creation -- emphasizing continuous delivery -- without creating more stress in the software development lifecycle (SDLC). It has become popular among teams also practicing Lean software development.

Kanban uses three basic principles: visualize the workflow; limit the amount of work in progress; and improve the flow of work. Similar to the Scrum, the Kanban method is

designed to help teams work more efficiently with each other. It encourages continuous collaboration and attempts to define the best possible workflow in order to promote an environment with active and ongoing learning and improvement.

The **Dynamic systems development method (DSDM)** is a response to the need for a common industry framework for rapid software delivery. The DSDM is based on eight key principles; failing to abide by any one of the principles introduces risk to successful completion of the project. The eight principles are:

1. Collaboration
2. On-time delivery
3. Demonstrated control
4. Continuous, clear communication
5. A constant focus on the business need
6. Iterative development
7. Creation in increments from firm foundations
8. Refusal to compromise quality

In the DSDM, rework is built into the process and all changes must be reversible. System requirements are prioritized using MoSCoW Rules, which ranks priority as:

- M -- must have
- S -- should have
- C -- could have, but not critical
- W -- won't have now, but could have later

It is important to the DSDM that not every requirement is considered critical. Each iteration should include less critical items which can be removed so higher priority requirements are not impacted.

Finally, **feature-driven development (FDD)** blends software engineering best practices -- such as developing by feature, code ownership and domain object modelling -- to create a cohesive, model-driven, short-iteration process. FFD begins by defining an overall model shape, which in turn creates a feature list. The method then proceeds with iterations that last two weeks and focus on planning by feature, designing by feature and building by feature. If a feature takes more than two weeks to build, then it should be broken down into smaller features. The primary advantage of FDD is that it is scalable -- even to large teams -- since it uses the concept of "just enough design initially," or JEDI.

Traditional vs Agile

Traditional Software Development:
Traditional software development is the software development process used to design and develop the simple software. It is basically used when the security and many other factors of the software are not much important. It is used by freshers in order to develop the software. It consists of five phases:

1. Requirement’s analysis
2. Design
3. Implementation

4. Coding and Testing
5. Maintenance

Agile Software Development:
Agile software development is the software development process used to design complicated software. It is basically used when the software is quite sensitive and complicated. It is used when security is much important. It is used by professionals in order to develop the software. It consists of three phases:

1. Project initiation
2. Sprint planning
3. Demos

Traditional Software Development	Agile Software Development
It is used to develop the simple software.	It is used to develop the complicated software.
In this methodology, testing is done once the development phase is totally completed.	In this methodology, testing and development processes are performed concurrently.
It provides less security.	It provides high security.
It provides less functionality in the software.	It provides all the functionality needed by the users.
It is basically used by freshers.	It is used by professionals.
Development cost is less using this methodology.	Development cost is high using this methodology.
It majorly consists of five phases.	It consists only three phases
It is less used by software development firms.	It is normally used by software development firms.