**IQRA NATIONAL UNIVERSITY PESHAWAR**

**Subject: Software Design**

Assignment

**Submitted By**

**Name :** Muhammad khan

**ID** : 15452

**Program** : MS(SE)

**To**

* **Dr.Fazal-e-malik saib**

**Introduction To Software Designing**

Selecting the right software development methodology for your product organization depends largely on your team size, goals, and other factors. Here is an overview of the most widely utilized and recognized software development methodologies to help you decide which is right for your team.

Software design methodology provides a logical and systematic means of proceeding with the design process as well as a set of guidelines for decision-making. The desi gn methodology provides a sequence of activities, and often uses a set of notations or diagrams.Software design is the process of envisioning and defining software solutions to one or more sets of problems. One of the main components of software design is the [software requirements analysis](https://en.wikipedia.org/wiki/Software_requirements_analysis%22%20%5Co%20%22Software%20requirements%20analysis) (SRA). SRA is a part of the [software development process](https://en.wikipedia.org/wiki/Software_development_process%22%20%5Co%20%22Software%20development%20process) that lists [specifications](https://en.wikipedia.org/wiki/Specifications%22%20%5Co%20%22Specifications) used in [software engineering](https://en.wikipedia.org/wiki/Software_engineering%22%20%5Co%20%22Software%20engineering). If the software is "semi-automated" or [user centered](https://en.wikipedia.org/wiki/User_centered_design%22%20%5Co%20%22User%20centered%20design), software design may involve [user experience design](https://en.wikipedia.org/wiki/User_experience_design%22%20%5Co%20%22User%20experience%20design) yielding a [storyboard](https://en.wikipedia.org/wiki/Storyboard%22%20%5Co%20%22Storyboard) to help determine those specifications. If the software is completely [automated](https://en.wikipedia.org/wiki/Automation%22%20%5Co%20%22Automation) (meaning no [user](https://en.wikipedia.org/wiki/User_%28computing%29%22%20%5Co%20%22User%20%28computing%29) or [user interface](https://en.wikipedia.org/wiki/User_interface%22%20%5Co%20%22User%20interface)), a software design may be as simple as a [flow chart](https://en.wikipedia.org/wiki/Flow_chart%22%20%5Co%20%22Flow%20chart) or text describing a planned sequence of events. There are also semi-standard methods like [Unified Modeling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language%22%20%5Co%20%22Unified%20Modeling%20Language) and [Fundamental modeling concepts](https://en.wikipedia.org/wiki/Fundamental_modeling_concepts%22%20%5Co%20%22Fundamental%20modeling%20concepts). In either case, some [documentation](https://en.wikipedia.org/wiki/Documentation%22%20%5Co%20%22Documentation) of the plan is usually the product of the design. Furthermore, a software design may be [platform-independent](https://en.wikipedia.org/wiki/Platform-independent_model%22%20%5Co%20%22Platform-independent%20model) or [platform-specific](https://en.wikipedia.org/wiki/Platform-specific_model%22%20%5Co%20%22Platform-specific%20model), depending upon the availability of the technology used for the design.

The main difference between software analysis and design is that the output of a software analysis consists of smaller problems to solve. Additionally, the analysis should not be designed very differently across different team members or groups. In contrast, the design focuses on capabilities, and thus multiple designs for the same problem can and will exist. Depending on the environment, the design often varies, whether it is created from reliable [frameworks](https://en.wikipedia.org/wiki/Software_framework%22%20%5Co%20%22Software%20framework) or implemented with suitable [design patterns](https://en.wikipedia.org/wiki/Design_patterns%22%20%5Co%20%22Design%20patterns). Design examples include operation systems, webpages, mobile devices or even the new cloud computing paradigm.

Software design is both a process and a model. The design process is a sequence of steps that enables the designer to describe all aspects of the software for building. Creative skill, past experience, a sense of what makes "good" software, and an overall commitment to quality are examples of critical success factors for a competent design. It is important to note, however, that the design process is not always a straightforward procedure; the design model can be compared to an architect's plans for a house. It begins by representing the totality of the thing that is to be built (e.g., a three-dimensional rendering of the house); slowly, the thing is refined to provide guidance for constructing each detail (e.g., the plumbing lay). Similarly, the design model that is created for software provides a variety of different views of the computer software. Basic design principles enable the software engineer to navigate the design process.

**Design considerations**

There are many aspects to consider in the design of a piece of software. The importance of each consideration should reflect the goals and expectations that the software is being created to meet. Some of these aspects are:

**Compatibility -** The software is able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.

**[Extensibility](https://en.wikipedia.org/wiki/Extensibility%22%20%5Co%20%22Extensibility) -** New capabilities can be added to the software without major changes to the underlying architecture.

**[Modularity](https://en.wikipedia.org/wiki/Modularity%22%20%5Co%20%22Modularity) -** the resulting software comprises well defined, independent components which leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.

**[Fault-tolerance](https://en.wikipedia.org/wiki/Fault-tolerance%22%20%5Co%20%22Fault-tolerance) -** The software is resistant to and able to recover from component failure.

**[Maintainability](https://en.wikipedia.org/wiki/Maintainability%22%20%5Co%20%22Maintainability) -** A measure of how easily bug fixes or functional modifications can be accomplished. High maintainability can be the product of modularity and extensibility.

Reliability ([Software durability](https://en.wikipedia.org/wiki/Software_durability%22%20%5Co%20%22Software%20durability)) - The software is able to perform a required function under stated conditions for a specified period of time.

**[Reusability](https://en.wikipedia.org/wiki/Reusability%22%20%5Co%20%22Reusability) -** The ability to use some or all of the aspects of the preexisting software in other projects with little to no modification.

**[Robustness](https://en.wikipedia.org/wiki/Fault-tolerant_system%22%20%5Co%20%22Fault-tolerant%20system) -** The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with resilience to low memory conditions.

**[Security](https://en.wikipedia.org/wiki/Computer_security%22%20%5Co%20%22Computer%20security) -** The software is able to withstand and resist hostile acts and influences.

**[Usability](https://en.wikipedia.org/wiki/Usability%22%20%5Co%20%22Usability) -** The software [user interface](https://en.wikipedia.org/wiki/User_interface%22%20%5Co%20%22User%20interface) must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for the majority of the users.[[6]](https://en.wikipedia.org/wiki/Software_design%22%20%5Cl%20%22cite_note-6)

**[Performance](https://en.wikipedia.org/wiki/Performance%22%20%5Co%20%22Performance) -** The software performs its tasks within a time-frame that is acceptable for the user, and does not require too much memory.

**[Portability](https://en.wikipedia.org/wiki/Software_portability%22%20%5Co%20%22Software%20portability) -** The software should be usable across a number of different conditions and environments.

**[Scalability](https://en.wikipedia.org/wiki/Scalability%22%20%5Co%20%22Scalability)**- The software adapts well to increasing data or added features or number of users.

**1. WATERFALL**

When it comes to software development, Waterfall is the most traditional and sequential choice. Although it’s usually viewed as an ”old school” or outdated method, it’s helpful to understand the history and structure of Waterfall to better appreciate the flexibility of more modern methodologies. First created in 1970, Waterfall was one of the most prominent methodologies for several decades because of its plan-driven approach.

Waterfall requires plenty of structure and documentation up front. It is divided into self-contained stages or steps. The first stage is vital, requiring a full understanding by both developers and customers of the project’s demands and scope before anything begins. The stages are relatively rigid and often follow this sequence: determine the project’s requirements and scope, analyze those requirements, design, implement, test, deploy and finally, maintain.

There’s a lack of flexibility with this approach, meaning what is decided by the customer and developer at the beginning must be seen through. Should any changes need to be made or mistakes addressed toward the end stages, the Waterfall method generally requires a full restart.

Typically, one stage must be finished before the next can begin, which can help with organization and assignments. And because the full scope of the project is understood in advance, software progress can easily be measured. Waterfall is often utilized by large, plan-driven teams who have a very clear understanding of the project’s scope;—however, development teams who don’t operate in a vacuum will likely find better results with the flexibility and agility of more modern methodologies.

**2. FEATURE-DRIVEN DEVELOPMENT**

An iterative and incremental approach to software development, Feature-Driven Development

(FDD) is derived from the Agile methodology and is considered one way to implement it. Similar to Waterfall, FDD is typically viewed as an older methodology, a sort of precursor to modern Lean/Agile implementations. FDD still focuses on the goal of delivering working software frequently and is an especially client-centric approach, making it a good fit for smaller development teams.

Features are a foundational piece of FDD. Features are client-valued pieces of work that, according to the FDD approach, should be delivered every two weeks.

To produce tangible software often and efficiently, FDD has five steps, the first of which is to develop an overall model. Next, build a feature list and then plan by each feature. The final two steps—design by feature and build by feature—will take up the majority of the effort. At each step, status reporting is encouraged and helps to track progress, results, and possible errors. Although efficient response to change is one of FDD’s better attributes, an understanding of the client’s requirements and the overall model at the beginning of the project can reduce any surprises during development.

Additionally, any feature that takes longer than two weeks to design and build must be further broken down into separate features until it meets the two-week rule. The rigid structure of FDD make it less desirable to teams who balance project-driven and break-fix types of work.

**3. AGILE**

The Agile methodology was developed as a response to growing frustrations with Waterfall and other highly structured, inflexible methodologies. This approach is designed to accommodate change and the need to produce software faster.

Agile values individuals and their relationships and interactions over tools; it features customer collaboration throughout the development process; it responds to change instead of following a set-in-stone plan; and it focuses on presenting working software, rather than documentation.

Unlike Waterfall, Agile is well equipped to handle the complexity and variability involved in development projects.Using the Agile approach, teams develop in short sprints or iterations, each of which includes a defined duration and list of deliverables, but in no particular order. During sprints, teams work towards the goal of delivering working software (or some other tangible, testable output).

Agile is collaboration-heavy, focusing on team strengths and efficiency, along with internal feedback from various departments and clients. Client satisfaction is the highest priority with the Agile approach, which teams achieve by continuously delivering working, tested, prioritized features.

**4. SCRUM**

Another way to implement the Agile approach, Scrum borrows from Agile’s foundational beliefs and philosophy that teams and developers should collaborate heavily and daily.

With Scrum, software is developed using an iterative approach in which the team is front and center—experienced and disciplined workers on smaller teams might find the most success with this method, as it requires self-organization and self-management.

Team members break down end goals into smaller goals at the beginning and work through them using fixed-length iterations—or sprints—to build software and showcase it often (which usually last two weeks). Meetings play an important role in the Scrum approach, and during each sprint, daily planning meetings and demos take place to follow progress and gather feedback. This incremental method promotes quick changes and development and adds value to complex projects. Scrum incorporates the structure and discipline of more traditional software development methodologies with the flexibility and iterative practices of modern Agile.

**5. EXTREME PROGRAMMING**

Another Agile framework, Extreme Programming (or XP) focuses on producing higher quality software using the best practices in software development. As with most Agile approaches, XP allows for frequent releases in short development sprints that encourage change when needed.

In general, XP follows a set of values, rather than steps, including simplicity (develop what is required, nothing more); communication (teams must collaborate and work together on every piece of the software); consistent feedback; and respect.

Extreme Programing requires developers to first plan and understand the customer’s user stories—their informal descriptions of certain features. Other practices include: scheduling and dividing work into iterations. Design with simplicity in mind, code and test often, which helps to create fault-free software. Listen to feedback to best understand the functionality, and then test more.

**6. LEAN**

Lean is at once a workflow methodology and a mindset, incorporating principles and practices from the manufacturing space and applying them broadly to a variety of industries, including software development. While Agile is an excellent methodology for the practical application of development best practices, it does not include instructions for scaling these practices across the organization or applying them outside of development-type work.

This is why many organizations who practice Agile at the team level begin to incorporate Lean philosophies, practices, and tools to help to innovate at scale. Lean’s basic principles—optimize the whole, eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, and respect people—can help to guide decision-making across the organization in a way that can help to unearth potential issues and maintain a healthy organizational culture.

Combining the best of Lean thinking and Agile software development practices can create a healthy, sustainable culture of innovation that benefits not only the development organization, but the system as a whole.

You can learn more about combining Lean and Agile in the following resources:

**Find to Chose Waterfall development method**

Many consider the waterfall method to be the most traditional software development method. The waterfall method is a rigid linear model that consists of sequential phases (requirements, design, implementation, verification, maintenance) focusing on distinct goals. Each phase must be 100% complete before the next phase can start. There’s usually no process for going back to modify the project or direction.

Pros: The linear nature of the waterfall development method makes it easy to understand and manage. Projects with clear objectives and stable requirements can best use the waterfall method. Less experienced project managers and project teams, as well as teams whose composition changes frequently, may benefit the most from using the waterfall development methodology.

Cons: The waterfall development method is often slow and costly due to its rigid structure and tight controls. These drawbacks can lead waterfall method users to explore other software development methodologies.