Analysis of Software Development Process Models: Comparative Study & Impact Study

Mrs. Purvi Sankhe
Research Scholar
Sanjeev Agrawal Global Educational University Bhopal, Madhya Pradesh

Dr. Mukesh Dixit
Associate Professor
Sanjeev Agrawal Global Educational University Bhopal, Madhya Pradesh.

ABSTRACT

The Software Development Life Cycle (SDLC) aims to plan, develop, create, test, and deliver high-quality software at the lowest cost and, ideally, in the shortest amount of time. This is only possible if the software engineering team chooses a software development model that is suited to the organization's requirements, the stakeholders' objectives, and the characteristics of the product. There are many different software development models, each with their own advantages and disadvantages.

Constraints on time and money are just two of the many things that should influence your model choice. Choosing the appropriate software development paradigm is crucial to a project's success. If the incorrect model is adopted, the project's timeframe will be extended, its prices will increase, its quality will deteriorate, and it may even fail.

The Software programs has become an integral part of our daily lives. There is a pressing need to establish a reliable process for creating software. These days, practitioners can choose between the heavy weight and light weight approaches. Waterfall, agile, and spiral are only few of the development models used by both approaches. The study compares and contrasts the two primary approaches taken by the software industry. The paper lays out guidelines for determining which model is most suited for a variety of uses.

Keywords: Software Development Models, Agile Process, High and Light weight models

1. INTRODUCTION

Software development life cycle (SDLC) models offer direction for handling the difficult work of building software. The success of your project in terms of quality, timeliness, cost, and the ability to meet stakeholders' expectations depends on the model you choose. There are currently more than fifty different SDLC models in use. There is no one best way for developing software; each offers pros and cons that may be pertinent to a particular project or team.

Software has played a significant role in our lives for the last forty years. Similar duties are carried out by numerous programmes in the banking, transportation, communication, marketing, finance, entertainment, and other sectors. The term "life cycle" describes the procedure used to create software. The Software Development Life Cycle (SDLC) is the best method for developing and updating software [1]. According to David Whitgift [2], initial code development and debugging were done side by side. The code and debug strategy quickly ran into problems when developing software for large and complex systems, such those used in aerospace.

Due to the spiral model's intricacy, Winston Royce proposed the waterfall model [3] as a substitute in 1970. This paradigm insisted on thorough requirements, documentation, and design prior to software development. The waterfall life cycle consists of a series of steps, starting with the requirement phase and continuing through testing and deployment. According to Blum [4], one of the waterfall model's shortcomings is that it is designed with hardware manufacture in mind while ignoring the special characteristics of software. The waterfall methodology is recommended for non-user-focused systems and software, such as operating systems and compilers, by Barry Boehm [5]. According to Glass [6], 29% of initiatives using the waterfall methodology failed.

The developers came to the realisation that they needed to immediately abandon the sequential waterfall method that it represented in favour of other models where users actively participate in defining requirements. According to Vector [7], waterfall is significantly hampered by uncertain, changing, and new needs. He continued by saying that because the feedback loop is shorter for items produced in smaller quantities, they are of higher quality and effectiveness.

Another the shortcomings of the waterfall method of software development have led to the creation of substitute methods. For instance, the spiral [9] is constructed to handle potential dangers that may arise during software development, and the prototype model [8] is created to assist in the clarity of requirements. David Whitgift [2] advises considering building it in parts if it's too unsafe to construct it all at once. The incremental model [10] is a method for managing complicated systems in the future.

In order to reach a final deliverable, these models allow iterative trips through the waterfall's iteration cycle. Software development approaches like waterfall, incremental, and spiral are referred to as "heavyweight methodologies" [8]. A methodology is created by combining one of the software development models with one or more techniques [11]. Prototyping and object-oriented approaches can be used to implement the waterfall, incremental, and spiral models. These techniques could be applied to one project. Scientists and programmers have been searching for innovative strategies for expediting software development due to time restrictions. The Rapid Development Model (RAD) has been introduced [12]. Prototyping and RAD models are examples of lightweight methodologies [8].

The Agile technique [13], a final lightweight strategy, was introduced. Iterative and incremental development, in which the stages are continuously reviewed, is the method used by agile development. Iterative software development is based on customer feedback.

One of the numerous techniques (models) under the Agile methodology that focuses on adapting requirements changes and cutting down development cycles is Extreme Programming (XP), which includes Scrum. "The agile is simply the latest theory," according to Victor [7] when contrasted to the waterfall approach.

The plan for this essay is provided below. The reasons why software projects fail will be covered in Section II, and the specifics of each model of the laborious technique, including advantages and disadvantages, will be covered in Section III. In Section IV, each model of the light methodology is discussed along with advantages and disadvantages. We compare the heavy and light approaches in Section V, and in Section VI, we go over the factors to take into account when choosing a model. The conclusion is presented in Section VII.

2. SOFTWARE FAILURE CAUSES

Over the past fifty years, there have been a number of setbacks in the development of software. Today, many people are still familiar with the term "software engineering," which was developed in the 1960s as a response to the frequent failure of software initiatives. Additionally, the millennium crisis of 2000 encouraged specialists to look for novel ways to resolve software bugs.

There are still problems with software development as demonstrated by the fact that even Microsoft has a history of delivering late and unsuccessful programmes (like Windows 1.0 and Windows 95).

We all have the same objective of seeing software ventures succeed. For software to succeed, the following qualities are necessary:

- User input
- Stakeholder satisfaction
- Management support
- A well-defined scope
- The ability to adapt to new circumstances
- A well-thought-out plan
- Close monitoring
- Expert project management
- The right tools and techniques

What causes so many software development projects to fail? Is the one we should be asking ourselves? Among the most common symptoms are:

The wrong approach was taken, and requirements weren't clearly defined. Market competition impossible project expectations Failure to handle risks Poor interaction between stakeholders Mismanagement of the project lacklustre project management The Standish Chaos Reports [14] are the go-to source for data on unsuccessful software initiatives. Successful software development projects will have all of the above characteristics. Reports of software failure [15] indicate 62 % Failed to meet schedule, 49 % Budget overruns, 41 % Failed to deliver ROI as per TCS.

3. SOFTWARE DEVELOPMENT MODELS

The phases of analysis, design, implementation, and testing form the backbone of heavy weight methodologies.
Documentation, long-term planning, and design are given primary importance in the Heavy technique. Here, we introduce some of the most prominent models associated with the heavy approach.

Keep in mind that development models have a significant impact on the project's outcome in terms of quality, cost, schedule, and stakeholder satisfaction.

1. Waterfall Model

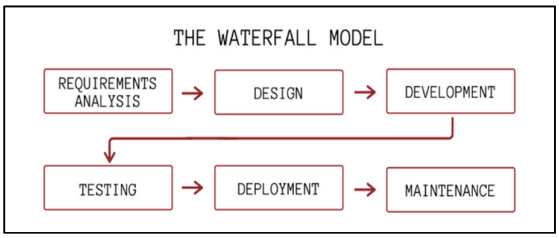

Figure 1: Waterfall Model

The original method for creating software was called the Waterfall model. The name of the process implies a sequential progression through the following steps of the software development life cycle: analysis, design, development, testing, deployment, and maintenance. There are clear milestones and deliverables at each juncture.

Comparing the Pros and Cons of a Waterfall Approach
In the Waterfall paradigm, one stage cannot proceed until the one before it is finished. When all of the milestone conditions are completed and the next phase of the project is approved, we say that the stage is complete.
Because you can't go backwards, overlap, or skip steps in this paradigm, it's quite rigid. This makes the process of modifying both time-consuming and costly. No matter when problems are discovered, they will not be addressed until the maintenance phase.

Due to its inflexibility, this strategy is more expensive and time-consuming than alternatives. This strategy carries a high degree of danger if requirements are ambiguous or misunderstood.

When additional adaptability is needed, such as in large, complex, or continuous projects, the Waterfall approach is not the best choice.

Despite its obvious drawbacks, this model's ease of use and rapid configuration make it a viable option for short-term, low-volume projects with flexible budgets and schedules. The group needs to make sure all requirements are crystal clear and immutable.

The benefits of the Waterfall model are outweighed by the drawbacks. Thus, this methodology has fallen out of favour as IT teams adopt agile software development, which promotes greater adaptability, better communication, and ongoing enhancements.

2. V-model (Validation and Verification model)

The V-model, also known as the Validation and Verification model, is an extension of the Waterfall methodology that emphasises forethought for test execution. When developing software, the V-Model follows a downward trajectory until the coding phase, where it turns a 90-degree corner and climbs upward in a "V" shape.

The Pros and Cons of the V-Model

There is testing involved at each phase of development. The team can catch mistakes in the project's requirement specs, code, and architecture at an early stage. As there is now no obvious solution to these issues, correcting them remains a challenging and expensive endeavour.
The V-Model has better odds of success than the Waterfall model because of the integration of early test planning. The V-Model, however, remains linear and thus rigid.
The team can only move on to the next level once the previous stage has been completed, much as in the Waterfall model. Because of this, alterations are cumbersome, pricey, and time-consuming.

This paradigm works well for small projects with well-defined needs that can be easily documented, but it struggles with more involved or longer-running projects.

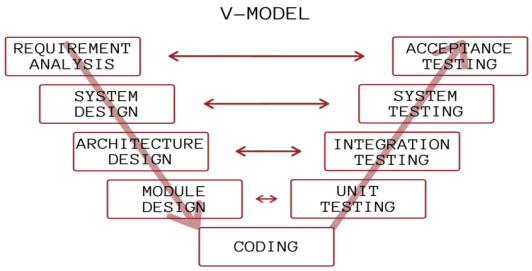

Figure 2: V Model (Validation and Verification model)

3. Incremental (Iterative) Model

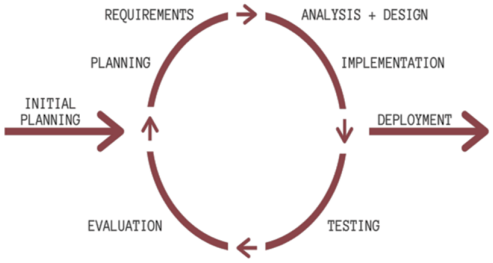

Figure 3: Incremental Model

The Iterative (and Incremental) model, like many others, was created to address the inadequacies of the Waterfall approach. Planning comes first, and then deployment, just like the Waterfall paradigm. This methodology differs from the Waterfall approach in that it uses iterative cycles.

These cycles, as the model's name implies, are iterative (repeated) and incremental (occurring over very brief time spans).

A minimal set of requirements is established at the outset of software development, and additional needs are added with each iteration.

Pros and Cons of an Iterative Approach

The model's iterative structure enables the programme to undergo continuous evolution and growth by permitting incremental improvements at each stage. Changes can be made by developers in light of what they've learned from past iterations.

Since not all project needs are known at the outset and many adjustments are made as the project progresses, it is possible to get started right away. Repeating the process regularly, however, can quickly use up resources and make administration more difficult.

However, this paradigm is still made up of established processes, which might lead to inflexibility at times, even though it does accommodate some change requests. In projects where needs are likely to change during the iteration, this approach is not optimal despite its lower change cost compared to the Waterfall and V-models.

Frequent modifications, undetermined costs and resource needs, and hazy deadlines are all potential dangers in an Iterative approach.

4. Prototyping Model

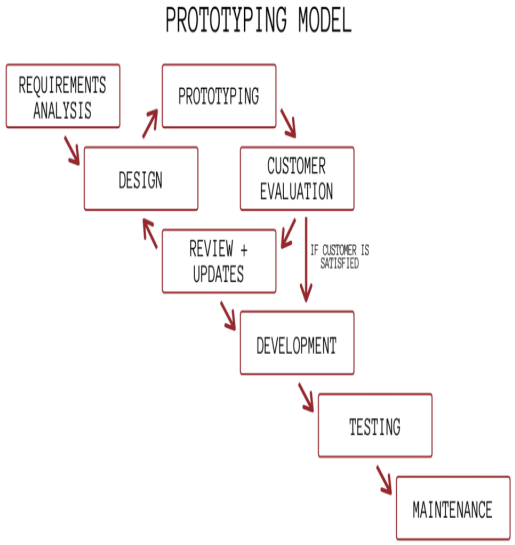

Figure 4: Prototyping Model

The main objective of the prototyping strategy is to aid the development team in better understanding the needs and wants of the customer. By creating a scale model of the finished product, misunderstandings and discrepancies can be worked out before full production gets underway.

The first thing developers do is to create a prototype to make sure they're on the right track with the customer's desires. The prototype is enhanced and developed further in response to the findings of the tests. When the prototype is accepted, the team gets to work on the finished product.

The Benefits and Drawbacks of Prototyping Techniques

Early user feedback obtained through the prototyping paradigm can drastically reduce the number of iterations required during the SDLC. This saves time and increases the likelihood that customers will be pleased.

The cost of the developer's effort in producing the prototypes, however, may outweigh the time savings. If the client requests multiple revisions, frequently changes their mind, or makes unreasonable demands, this time can quickly mount up. This is why it's advised to set a limit on how many alterations can be made before the prototype must be approved.

Once work on the final prototype has started, no additional requirements or modifications to the plan can be made. The prototype approach has a significant problem.

5. Spiral Model

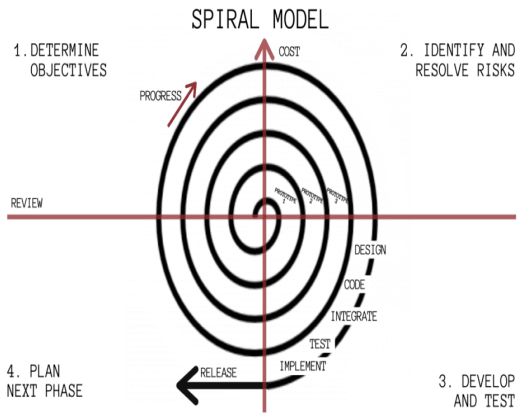

Figure 5: Spiral Model

Potential threats are assessed using the spiral model. Any team using this methodology must therefore be comprised of subject-matter specialists.

These four processes—planning, risk assessment, design, and assessment—combine to form this methodology. The spiral's reclusiveness depends on the project's scope and the project manager's discretion. Software development normally takes this process about six months.

It incorporates aspects of both those strategies by putting an emphasis on design, which involves prototyping (during the engineering phase), and by adhering to phases like those in the Waterfall model.

The Spiral Model's Benefits and Drawbacks

Continuous and repeated development allows developers to make changes and add new features while reducing risks. Waste is also decreased in part by the development process's structured character.

If the client is slow to provide feedback at any point in the cycle, it could be problematic for the development process. However, customer adjustments are not allowed during the engineering process.

The number of loops, or iterations, is unpredictable, so there is a risk of going over budget and missing deadlines. Most of the time, it costs a lot of money to get the desired outcome.

Additionally, because the model is so specifically adapted to each client, it is impossible to reuse your work.

6. About Agile Methodology

The 12 principles of the Agile Manifesto serve as the foundation for the practises that make up agile. It is a method of thinking rather than a strict set of guidelines.

Agile was developed to produce better software more effectively and efficiently than prior methodologies, such as Waterfall, by emphasising cooperation, communication, and quick change.

Agile was quickly and broadly embraced. According to the Project Management Institute, 71% of firms indicate they occasionally, frequently, or usually use agile approaches for projects.

There are various Agile software development models. These techniques are fundamentally based on collaboration, cross-departmental coordination, gradual progress, and quick responsiveness to customer feedback. Teams that use testing, feedback, and iteration produce and deliver the best software. Think about two different agile software development strategies.

6.1. Scrum Model

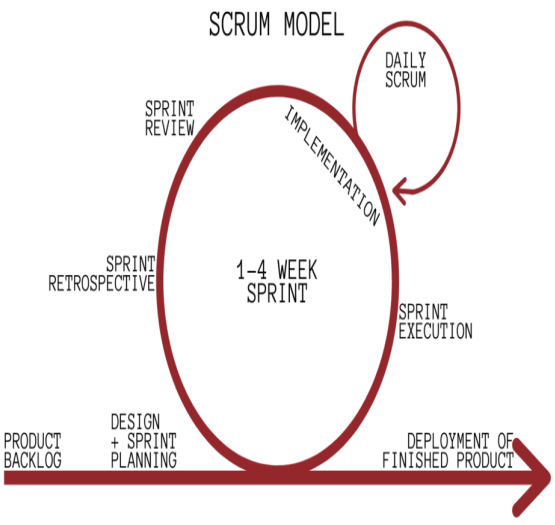

Figure 6: Scrum Model

The most widely used agile model is the Scrum one. Sprints are the name for its iterations of software development. Teams evaluate the preceding sprint, add new features (functionality that has been created and tested), and plan the following sprint during these 1-4 week sprints. After the sprint's activities are established, changes are not permitted.

New features and items are introduced at the end of each sprint to be coded and tested in the following sprint. This continues up until the project is declared ready for release and all features have been incorporated.

Advantages and disadvantages of the Scrum Model

The amount of guessing and errors that frequently occur from inadequate communication are decreased through increased collaboration between cross-functional teams and between the company and the client. The additional steps also shorten the time to market.

Improved communication decreases the amount of time needed to fix faults and enhances the possibility that the product will be liked by the customer. The consumer will need to spend more time and provide more input throughout this cooperation, though. The team runs the danger of missing deadlines by adding and requesting too many features.

6.2. Kanban Model

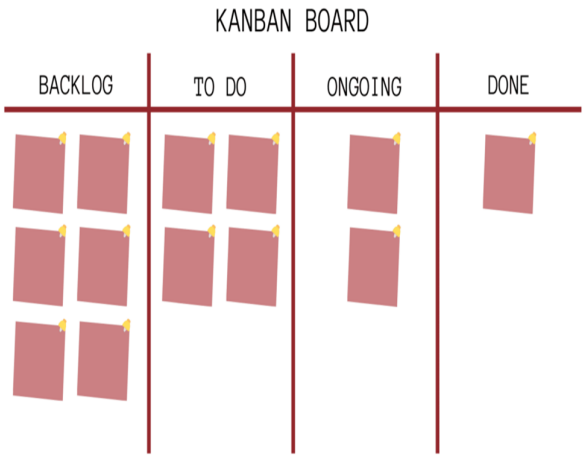

Figure 7: Kanban board

Kanban does not have pronounced iterations like the other models do. If a team does schedule iterations, they are usually very brief sprints, sometimes no longer than a single day.

Sticky notes are used to visually outline the projects and their information, such as project owners and progress status, on a Kanban board. The team can focus their attention more narrowly on the most crucial feature that is currently under development thanks to this visualisation.

The Kanban board also emphasises the opportunity for further feature perfection through ongoing improvement.

Benefits and Drawbacks of the Kanban Model

Although using sticky notes on a board encourages the team to concentrate on improving the crucial task at hand, it is a bad technique to establish and uphold timetables. Long projects are therefore exceedingly difficult to plan.

There is no predetermined stage of planning, so adjustments can be made at any time. The absence of deadlines is a common drawback of Kanban. If modifications are made frequently, this problem may be made worse.

4. COMPARISION OF SOFTWARE DEVELOPMENT MODELS

It appears from the previews that heavy techniques focus on the people and the changes, whereas light methods focus on the processes, the plans, and the documents. These characteristics help us choose the appropriate software development approach for any specific project. Here are a few things to think about before deciding on a methodology for your next project. Teams using light technique should have no more than 10 members. On the other hand, projects requiring a large team can be managed with the help of heavy technique.

In contrast to the focus on working software by light methodology, heavy procedures rely heavily on documentation. When requirements are known in advance and stay constant throughout the life cycle, heavy weight techniques shine. Heavy procedures rely heavily on careful preparation. The plan for a light technique is informal and created quickly before each step.

Requirements change: While the heavy methodology opposes requirements change, the light methodology embraces it, and the cost of change is much lower, especially in later phases.

Whereas heavy procedures rely heavily on communication documents, light approaches rely on interaction between developers and users. There are many different models used in software development. In this section, we outline some of the most important factors considered while settling on the optimal model for software creation. The success or failure of software initiatives is commonly regarded to depend on several factors. Each model was developed to streamline and enhance some aspect of the software production cycle. As it is right now, each software development approach is optimal for certain kinds of projects. Manual processes like Waterfall, which have been around for a while, are, nevertheless, becoming increasingly obsolete.

IT departments and organisations as a whole need to improve their efficiency and velocity if they want to satisfy customers, meet deadlines, and release new software on time. Automation forms the backbone of a software development process that is faster, more repeatable, and more secure. Not all designs can be automated or made that quickly. This has led to the widespread adoption of the agile framework. Continuous integration, continuous delivery, release automation, and DevOps are just a few of the enabling technologies and procedures that businesses are implementing. This work also derive few factors most likely considered before selecting the software development models. Which are; Risk, Requirement, Frequency of change, Documentation, User Involvement, Cost & Delivery time.

5. EXPERIMENTAL ANALYSIS

The study of various software development models address that software development is not an simple or straight forward job which can be start by thinking about development and just end by execution. Here various stockholders are involved which need attention of several points. This research work perform the study on various software models by examining the developer view point and conclude the complete work in comparative analysis.

This research work start by collecting different opening from various software development firms on different points. A brief about data collection points are cited below;

1. Model name
2. Recommended Project Size
3. Documentations
4. Requirement Collection
5. Release Mode
6. Client Involvement
7. Team Size
8. Work Status Tracking
9. Client Happiness
10. Organization Size
11. Efforts
12. Project Completion Probability

The data collection has been done among 461 developers experienced from fresher to 10 years for more than 20+ project work experience. The complete data collection has end by observations which are shown in table 1 and figure 8 respectively.

Table 1: comparative analysis of software development models

Software Development Model	Waterfall	Iterative	Spiral	Agile
Recommended Project Size	Small	Large	Large	Large
Documentations	Detailed	Medium	Medium	Easy and Less
Requirement Collection	At time of inception only	After each iteration	After each cycle or release	
Release Mode	End of the model	Iteration wise	Cycle wise	Milestone wise[weekly/fortnight/Monthly]
Client Involvement	Low	Medium	Medium	High
Team Size	Large	Average	Large	Small

Work Status Tracking	Complicated	Average	Average	Easy
Client Happiness	Very Low	Vary-Project to project	Vary-Project to project	Very High
Organization Size	Small	Small-large	Small-large	Small-large
Efforts	Hectic	Moderate	Moderate	Simple
Project Completion Probability	Very Low	Low	Low	Very High

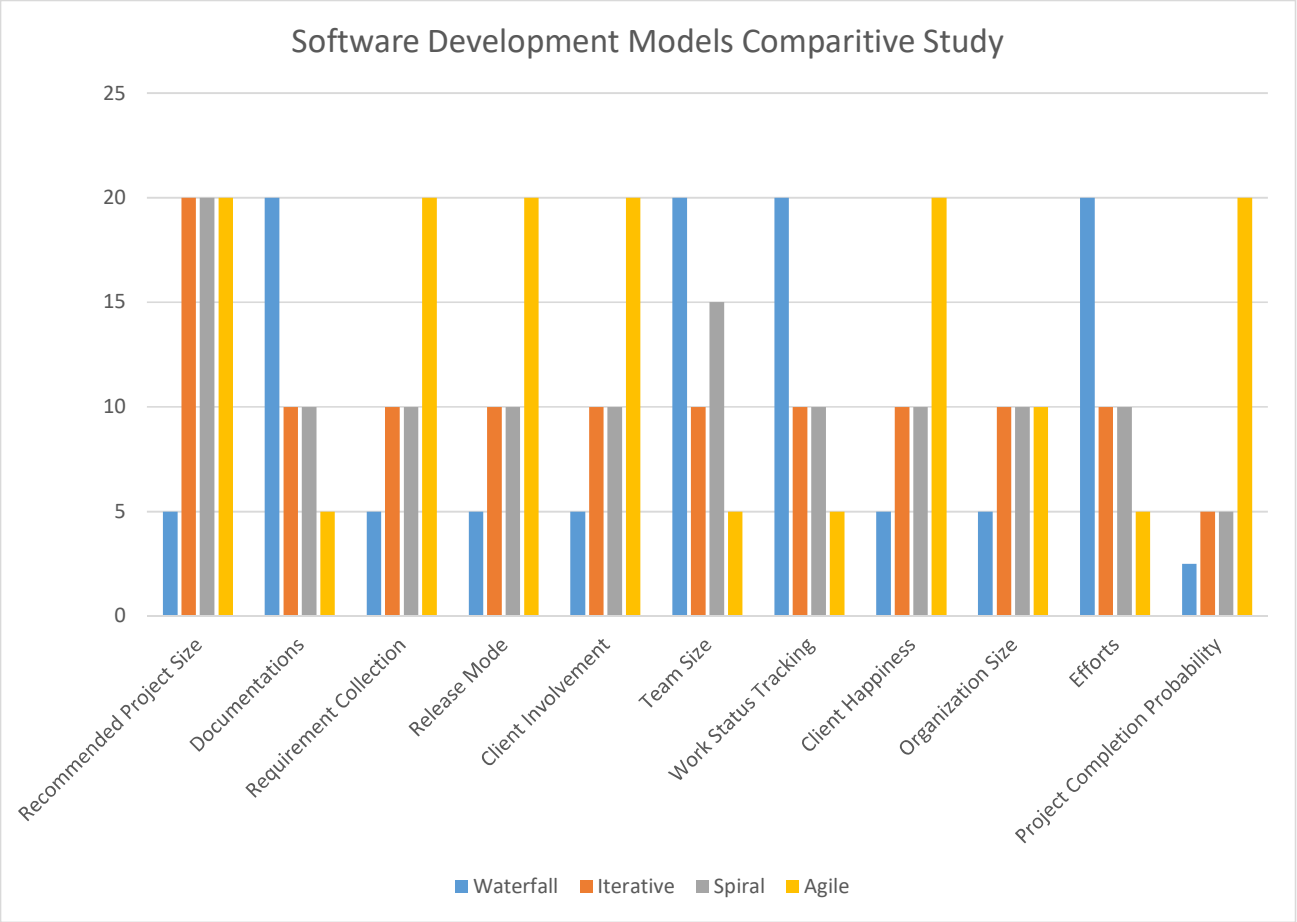

Figure 8: Software Development Model Comparative Study

13. CONCLUSION

We urgently need an appropriate system that will guarantee consumer satisfaction, manufacture high-quality items at fair costs, and easily adapt to a constantly changing environment. There is no assurance that either the heavy or light methodological models will be the best or worse option. Making the ideal model choice involves taking into account a number of factors, such as the nature of the project, the development team's skills, and senior management's support. Using the same or a different technique, models can frequently be combined with one another (hybrid model). The complete study concludes that waterfall is market leader of old time but agile has replaces almost all software development techniques from small to large size projects in all size organization. Waterfall only used by fixed bid projects having very short duration timeline. But for long term and scalable projects only agile could be a visionary solution.

REFERENCE

[1] Asif, Usman Khan and Rizwan Qurashi, “A comprehensive Study of Commonly Practiced Heavy and Light weight Software Methodologies”, USCI International Journal of Computer Science Issues, vol 8 , 2011

[2] Kerr JM, Hunter R. 1993. Inside RAD: How to Build a Fully-Functional System in 90 Days or Less. McGrawHill: Columbus, USA.

[3] Kroll P, Isaac BM. 2006. Agility and Discipline Made Easy: Practices from OpenUP and RUP. Addison-Wesley Professional: Boston, USA.

[4] Kroll P, Kruchten P. 2003. Rational Unified Process Made Easy, The: A Practitioner's Guide to the RUP. AddisonWesley Professional: Boston, USA.
[5] Larman C. 2003. Agile and Iterative Development. Addison Wesley: Boston, USA.
[6] Mangione C. 2003. Software Project Failure: The Reasons, The Costs. Available at: http://www.cioupdate.com/
[7] reports/article.php/1563701/Software-Project-FailureThe-Reasons-The-Costs.htm [21 November 2012].
[8] Martin RC. 2011. Agile Software Development, Principles, Patterns, and Practices. Pearson: London, UK.
[9] Mills H, Dyer M, Linger R. 1987. Cleanroom software engineering. IEEE Software 4(5): 19–25.
[10] Munassar NMA, Govardhan A. 2010. A comparison between five models of software engineering. International Journal of Computer Science 7(5): 94–101.
[11] Pichler R. 2010. Agile Product Management with Scrum: Creating Products That Customers Love. Addison Wesley: Boston, USA.
[12] Poppendieck M, Poppendieck T. 2003. Lean Software Development: An Agile Toolkit. Addison-Wesley Professional: Boston, USA.
[13] Prowell SJ, Trammell CJ, Linger RC, Poore JH. 1999.
[14] Cleanroom Software Engineering: Technology and Process. Addison-Wesley: Reading, Mass..
[15] Royce W. 1970. Managing the development of large software systems. Proceedings of IEEE WESCON, August; 1–9.
[16] Schwaber K. 2004. Agile Project Management with Scrum.
[17] Microsoft Press: Washington, USA.
[18] Sims C, Johnson HL. 2012. Scrum: A Breathtakingly Brief and Agile Introduction. Dymaxicon: Marston Gate, UK.
[19] Succi G, Marchesi M. 2001. Extreme Programming Examined. Pearson Education: Upper Saddle River, NJ, USA.
[20] Turner JR. 1993. The Handbook of Project-Based Management. McGraw-Hill: Maidenhead, UK.
[21] Z.Q. Lin, B. Xie and Y.Z. Zou, Intelligent Development Environment and Software Knowledge Graph, Journal of Computer Science and Technology, vol. 32, issue 2, pp.242-249, 2017.
[22] Chemsripong, Sujinda, and Chutchonook Charutwinyo."Competency Model of software Developer in Thailand." European Journal of Molecular & Clinical Medicine 7.3 (2020): 714-722S,
[23] Shylesh, A Study of Software Development Life Cycle Process Models (June 10, 2017). Available at SSRN: https://ssrn.com/abstract=2988291 or http://dx.doi.org/10.2139/ssrn.2988291
[24] Maryani Maryani , Hendra Alianto , Anzaludin Samsinga Perbangsa , Yulius Human Resources Management Information System Requirement Analysis and Development in Humanitarian Foundation , p. 284 Posted: 2023