International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 04 | April - 2023 Impact Factor: 8.176 ISSN: 2582-3930

Preempting Criteria Affecting Future Software Development Methodologies to Construct A Better SDLC Framework

Shreyansh Padarha[1]

[1] *Department of Data Science, CHRIST (Deemed To Be) University Pune Lavasa*

Abstract - Anthropological studies and system sciences show humans thrive under locus, where structural integrity is intact. It is only natural, that the software developed by humans, order the same. The process and methodology involved in building these applications are complex and chaotic, yet elegant at the same time. Dividing tasks into phases was considered the optimal method, to take advantage of the specializations of various individuals and teams. But, in the ever-rampant world, where AI and machine learning enable, concurrent implementation, it's just to question the basis of the same. The software development industry constantly evolves, with emerging new methodologies and technologies. To stay competitive and continuously evolve, better frameworks for future development and micromanaging need to be created. To do the same, this research paper aims to preempt the criteria that are likely to affect software development models and methodologies, by deep diving into the SLDC's origins, central idea, and relevance throughout our modern history.

***Key Words*:** Management, Software Development Life Cycle (SDLC), Software Engineering, Agile, Waterfall Models, Incremental Models, Generative AI, Artificial Intelligence, Blockchain Technology

INTRODUCTION

To construct a better SDLC framework, it is important to take a comprehensive approach that considers various factors such as project size, complexity, budget, time-to-market, regulatory compliance, and emerging technologies like Generative AI, Neural Networks, and Blockchain. The success of software development projects is heavily influenced by how people interact with technology, learn and adapt to new tools and processes, and communicate and collaborate with team members. Understanding human psychology and incorporating it into the design of SDLC frameworks and methodologies can create intuitive, efficient, and effective development processes. Collaboration, communication, and flexibility are also crucial for successful software development. By adopting new technologies and best practices, companies can proactively address the criteria affecting future software development methodologies and deliver high-quality software applications within the expected timeframes and budget.

To construct a better SDLC framework, it is important to take a comprehensive approach that considers various factors such as project size, complexity, budget, time-to-market, regulatory compliance, and emerging technologies like Generative AI, Neural Networks, and Blockchain. The success of software development projects is heavily influenced by how people interact with technology, how they learn and adapt to new tools and processes, and how they communicate and collaborate with team members. Understanding human psychology and incorporating it into the design of SDLC frameworks and methodologies can create intuitive, efficient, and effective development processes. Collaboration, communication, and flexibility are also crucial for successful software development. By adopting new technologies and best practices, companies can proactively address the criteria affecting future software development methodologies and deliver high-quality software applications within the expected timeframes and budget.

SDLC'S HISTORY, CLOUT & IMPORTANCE

Looking after Software development processes or methodology has been a norm in companies across various industries. The software development life cycle (SDLC) is an outline for developing software, it encompasses the various phases involved in a development process, from requirement gathering and designing to testing and maintenance. It is a systematic approach and a fundamental of software engineering. The process itself includes, but is not limited to planning, analysis, design, implementation, testing, deployment and maintenance. Each phase of the process is of critical importance.[1][2]

Existential need for SDLC and its history

In the early 1960s, structured programming and software engineering practices were coming into widespread usage. In the early days, software engineering, lacked consistency, formalization and any homogeneous structure, resulting in software products riffled with reliability and quality issues. In order to address these discrepancies, engineers started developing programming practices and methodologies that had a greater emphasis on requirement gathering, designing, testing and compartmentalised execution. The SDLC was developed as a formal framework for the development of software in an efficient, and quality manner.[3]

SDLCs' clout and widespread adoption

Software Development Life Cycle (SDLC) has been instrumental in the past few decades, and to understand its clout and significance, we'll be delving into in-depth case studies and the implications of using SDLCs for these undermentioned organisations, and industries.

A) Microsoft Windows (1990s): At the time, Microsoft was facing extremely tough competition from IBM's OS/2 and Apple's MacOS. Microsoft was faced with the challenge of developing a new OS. They used SDLC to manage the development of Windows, which ensured the project was developed on time and within the estimated budget. The SDLC also helped Microsoft ensure the quality of Windows OS was above par, which helped to establish Microsoft as the frontrunner in the modern era of Operating Systems. Although not confirmed, Microsoft most likely used an amended linear life cycle model, like the waterfall model[4][5][6][7]

B) Financial Industry: Companies like JPMorgan Chase, Goldman Sachs, and American Express make and other financial institutions make use of SDLC to comply with strict regulatory requirements such as Sarbanes-Oxley Acy and Dodd-Frank Act. These regulations require financial institutions to have strict procedures, and controls in place to ensure fairness, accuracy, completeness and reliability of financial data. The SDLC provides a framework that helps these organizations to comply with the requirements.[8][9]

C) ISO/IEC 12207: It is an international standard that provides a framework for software life cycle processes. The standard describes the processes that are necessary for the development, maintenance and disposal of all software products. The standard also highlights the importance and need to identify and manage risks associated with software development, whether it's security threats like data breaches or procedural/IP infringements. This standard shows the widespread adoption and validity of SDLC.[10][11]

D) IBM BENEFITING FROM SDLC APPROACH (360 OS)˘: In-depth stage-wise analysis in the next section.[12][13]

IBM BENEFITING FROM SDLC APPROACH (360 OS)

In 1964, International Business Machines Corporation (IBM) announced the development of the OS/ 360 operating system. The project, a massive undertaking, involved over 1,000 developers and cost over $ 1 billion. This project was considered a landmark in the history of software development and highlighted the vital need for a more structured and methodical approach to software development. IBM used the SLDC to manage the development of the OS/360, which helped ensure that such a mammoth project was completed in the given timeframe and budget.[12][13]

The SLDC fortified the OS/360 project, and helped IBM in the following ways:

1. Planning: In the planning phase, IBM defined rigid objectives and requirements. They identified the need to develop an OS that could support a wide range of computer models and configurations. The planning phase helped IBM to establish a clear project vision and roadmap.[14]

2. Analysing the requirements: IBM conducted extensive market research to identify the needs of its customers. The data and information gathered through this research helped define the functional requirement of the OS/360 operating system. This 'analysis' phase ensured the software being developed meets the needs of its target (end) users and provides them with the necessary functionalities.[15]

3. Design: IBM created a detailed design plan (strategy), which included the system architecture, data model and component design. For instance, IBM developed a hierarchical system structure, in order to allow for the efficient processing of huge amounts of data. This design phase meant that the product created was well-structured and could be maintained and scaled for the long term as well.[16]

4. Implementation: This phase involved creating the designed product. It involved coding, testing and integrating various components and parts of the system. IBM developed a set of programming languages that could be used to write programs for the OS/360. They also developed other utility tools to support the development and testing of these programs. This phase, helped IBM to fulfil and make a fully functional product that would meet the initial idea of the design and analysis phase.[17]

5. Testing: This phase helped IBM in rectifying bugs and make sure OS/360 was reliable for end users. The company conducted extensive testing of the system to identify and fix bugs and ensure the software ran across various computer configurations and models.[18]

6. Maintenance: The maintenance phase for a big company like IBM, is never-ending. This phase made sure that IBM kept OS/360 up to date and kept providing its users with ongoing support and improvements. For example, IBM included virtual memory management and the ability to support (compatible) multiple programming languages in its software.[19]

PARADIGM SHIFT IN MODEL ADOPTIONS

Context switching, in operating systems, refers to the component that allows multiple processors to share the same CPU. It allows the partially executed data to be stored until the processor is free to execute the remaining process. In contrast to the aforementioned phenomenon, in a Software Development Model or Methodology, once a model has been chosen to develop software, or complete a task chunk, changing the SDLC model can be extremely time-consuming, complex and challenging. In fact, companies as a whole avoid radically changing their methodologies on short notice, as it can create ripple effects on their operations. There are many

hindering factors to a change SDLC, like resistance to change, lack of skills and expertise, company culture and mindset and the obvious reason, Cost and time.[20]

Irrespective of this, adopting the wrong SDLC can be even more detrimental in both the short and long run. The consequences of selecting the wrong model can range from inefficiencies to delays, increased costs, etc. Over the past 2-3 decades companies have been shifting from traditional and old software development models to newer, hybrid, rigorous and bold models. This is largely credited to changing nature of the software development industry and the need to keep up with the rapidly evolving technologies and business requirements.[21]

The paradigm shift in model adoptions can be better understood with the classic waterfall to agile model shift. The waterfall model is the oldest, simplest model in existence. It's linear, and sequential and prevents re-accession of any phase. It's best suited for smaller projects, it's simple to understand and implement, systematic and structural in nature. Most major IT giants initially began with the waterfall model, which mostly got the job done, But, IBM. Intel and Microsoft, all have shifted from waterfall to agile in this century. Microsoft felt that the waterfall model was slowing down their development process, preventing them to meet customer expectations. IBM felt the model was too rigid and didn't allow changes to be made easily. Intel, on similar lines, felt the model was too inflexible and didn't welcome quick changes. Agile, on the other hand, allows for changes to be made quickly and efficiently resulting in faster delivery and more customizability of products.

Similar examples, of more companies, and their reasons to shift models have been drafted below.

Company	**Previous** SDLC Methodology	**Current** SDLC Methodology	**Reasons for Model Shift**
Google	Traditional SDLC Model	Hybrid Model (Agile + Scrum)	To enable faster development and release of products. Improve collaboration between teams. Increase productivity in product development.
Airbnb	Hybrid SDLC Model	RITE (Rapid Iterative Testing and Evaluation) Framework	To enable faster testing and iteration of products. Improve collaboration between teams. Improve the user experience.
Walmart	Traditional SDLC Model	Hybrid Model (Agile + DevOps)	To reduce costs. To increase speed to market rate. To improve product quality.
Cisco	Traditional SDLC Model	Continuous Delivery Methodology	For faster delivery of products. Improve collaboration between teams. Increase efficiency in the development process.
Microsoft	Waterfall Model	Agile Model	To speed up their development process. To be more responsive to customer needs.
IBM	Waterfall Model	Agile Model	To increase productivity To reduce development costs. Improve customer satisfaction
Intel	Waterfall Model	Agile Model	
Spotify	Scrum Model	Squad Model	To align their development process with their company culture

FACTORS INFLUENCING THE ADOPTION OF A METHODOLOGY

As seen in the previous section, there are various factors that come into play while choosing the SDLC model or methodology to use for developing products. A company or unit needs to keep these factors in mind. These criteria can also help us decide what kind of future framework or model could be adopted by companies and businesses.

In my opinion, the said factors can be broadly categorized under two related sections:

1. Organisation-bound factors

Organisational bound factors are those which are related to an individual company. It can range from the inner functioning of a company, and its domain to its work culture. Some of the important organization-specific factors include:

Culture and Mindset: The work culture and mindset of current employees at the company can deeply influence, whether they will adapt to a different methodology or model.

Resistance to change: The change in the model needs to be well received among the employees, in order to allow a smooth transition.

Lack of skills and expertise: This point can be better explained by an example, changing from a waterfall model to agile or scrum methodology, which involves being familiar with tools that clock in work, or bifurcate tasks. Employees need to either be trained or have prior knowledge/experience regarding these methodologies.

Technical Debt: This refers to the additional implied cost and effort required for the additional rework caused by choosing a fast, new technology as a solution, rather than the better approach that would have taken longer.

Other Factors: Cost and time, Customer Involvement

2. Model-instigated factors

Model-instigated factors include those factors that pertain to the complexities that a model/methodology being applied or adopted presents. Some of the prominent model-instigated factors involved:

Choosing the right model: depending on the projects on-board, size, requirements and complexity.

Stakeholder involvement: impact of the model on developers, testers, project managers and end-users.

Other Factors: Regulatory compliances in accordance with the model, Cost delta with new model

EMERGING TECHNOLOGIES AND CHANGING WORLD

Generative AI, Neural Networks, Blockchain and new technologies have changed the SDLC landscape and potential ground-breaking developments are a surety. Some imminent incorporation of new emerging technologies could be

a) Increased use of automated testing and deployment: Software development cycle models and methodologies will have to make use of AI, ML tools and frameworks for the development and testing of products to ensure faster development, implementation and deployment.

b) Designing and requirement analysis could be heeded with the incorporation of AI: Generative AI and complex ML models can render designs and possible requirements based on customer objectives and problem statements. There could be multiple renditions, that the AI produces and suggests, which can further be filtered and selected manually by experienced employees/managers/decision makers.

c) A shift to decentralized development: Blockchain technology can enable decentralized development and the creation of decentralized applications that operate without central control. As companies shift towards blockchain technology, SDLC might have to shift to similar development techniques.

d) More flexible and agile models: With AI there will be a greater emphasis on experimentation, iterations and feedback. SDLC models must be more flexible and agile to incorporate the variations, with an increased focus on collaboration and continuous improvement.

e) Greater emphasis on security and privacy: With blockchain technologies security and privacy will be of prime concern in the SLDC. New methodologies, will have to incorporate specialized security and privacy practices, such as smart contract development and testing.

CONCLUSION

The research paper found a great emphasis must be placed on how the inner workings of a company, might be influenced by the adoption of a new model of methodology. Over the years, the adoption of SDLC models has shifted significantly, dictating a paradigm shift. Factors influencing the adoption are not limited to a certain set of criteria. It can range from project size, complexity, budget, and time to market to regulatory compliance. To construct better SDLC frameworks for the future, it's imperative to take a multifaceted approach that considers various new emerging technologies like Generative AI, Neural Networks and Blockchain.

REFERENCES

[1] Boehm, B. W, *A spiral model of software development and enhancement. Computer, 21(5),* 1988

[2] *Selecting a Development Approach*, February 2005

[3] Geoffrey Elliott, *Global Business Information Technology: an integrated systems approach*, 2004 Brooks/Cole Publishing Company, 1993.

[4] A. Miller, *Applying the Systems Development Life Cycle to the Art of Writing Computer Programs. Journal of Information Technology Management*, 2001

[5] J. Stien, *Microsoft Windows prepares for its debut, Infoworld*, 1991

[6] Microsoft, *Windows 3.0*, 1990

[7] R. Magoulas, *Windows XP in a nutshell, O'Reilly Media Inc.*, 2004

[8] *Sarbanes-Oxley Act of 2002, Pub. L. No. 107-204, 116 Stat. 745*, 2002

[9] *Dodd-Frank Wall Street Reform and Consumer Protection Act, Pub. L. No. 111-203, 124 Stat. 1376*, 2010

[10] B. W. Boehm & L. B. Huang, *"ISO/IEC 12207: A Framework for Software Life Cycle Processes." Computer, 34(9), 112-113.*, 2001

[11] ISO/IEC 12207:2017, *Systems and software engineering – Software life cycle processes, International Organization for Standardization (ISO)*, 2017

[12] IBM Systems Reference Library, *IBM Operating System: Concepts and Failures*, 1965

[13] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Sixth Edition)*, 2005

[14] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Seventh Edition) Chapter 2*, 2010

[15] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Seventh Edition) Chapter 4*, 2010

[16] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Seventh Edition) Chapter 5*, 2010

[17] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Seventh Edition) Chapter 6*, 2010

[18] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Seventh Edition) Chapter 9*, 2010

[19] Roger S. Pressman, *Software Engineering: A Practitioner's Approach (Seventh Edition) Chapter 15*, 2010

[20] Anneliese Andrews & James Dalziel, *The impact of culture on software development: a systematic literature review, Journal of Software: Practice and Experience*, 2017

[21] S. S. Rathore & K. Sing , *Software Development Life Cycle Models and Methodologies: A Survey, International Journal of Advanced Research in Computer Science and Software Engineering*, 2016