**Abstract**

In this report we will discuss different stages of SDLC in traditional way. The architecture will be studied and we will be building a connection between SDLC and architecture. The discussion will show why it is important to reorient SDLC with respect to architecture. The further discussion will reveal how SDLC incorporated architecture in modern day software engineering practices and how it is taken care of during different phases of SDLC.

**Introduction**

In software engineering, architecture has a prime importance and is extensively involved in engineering process throughout the globe. In the initial days of software development architecture was not that much famous. The traditional understanding of SDLC were mostly without discussion of architecture. But now in modern day software engineering, the process is carried out by keeping development challenges, future plans, and factors like reusability, scalability and maintenance. All these factors causes the strong interference of software architecture in the field of software engineering. Almost every part and phase of software development involves architecture in some direct or indirect way. Since the architecture is so much involved in software engineering process, we need to elaborate the phases of SDLC to see the involvement of architecture at different stages of software development.

**SDLC at a glance**

**Architecture and its importance**

Software architecture is the foundation of a software system. Like other types of engineering, the foundation has a profound effect on the quality of what is built on top of it. As such, it holds a great deal of importance in terms of the successful development, and eventual maintenance, of the system.

The software architecture of a system depicts the system's organization or structure, and provides an explanation of how it behaves. A system represents the collection of components that accomplish a specific function or set of functions. A software architect makes important decisions regarding the software that goes on to define its overall integrity. A good software architecture helps define attributes such as performance, quality, scalability, maintainability, manageability, and usability.

**Why Does Software Architecture Matter?**

An organized software architecture helps to ensure the longevity of the software’s internal quality.

Consider two similar products. Both are launched within a month-long gap and aims to add new features when they complete three months.

There are two scenarios:

* Product A launched in September 2022. This project supports a messy source code because the [development team](https://www.netsolutions.com/hire-developers) wanted to launch and monopolize the market as early as possible.
* Product B launched in November 2022. This project has a software architecture that is well-structured and organized. The development team works on the design and architectural decisions early in the process and prioritizes quality over faster launch.

Which Product will be more successful: A or B?

Product A might monopolize the market initially and convert better. However, product adoption will eventually subside because the messy code will lead to [technical debt](https://www.netsolutions.com/insights/what-is-technical-debt/) pileups. These pileups will, in turn, make it challenging to introduce new updates and bug fixes on the fly.

Product B might have a market entry gap, but it will be easier to maintain a faster shipping cadence. The customer needs will be looked after without breaking the shipping cadence, thus making for a larger win.

## ****Why is software architecture important?****

A proper foundation laid down by a software system's architecture yields a number of benefits. Let's take a deeper look at those benefits:

### ***Defining a solution to meet requirements***

Software strives to meet all functional, non-functional, technical, and operational requirements. Working closely with stakeholders, such as domain experts, business analysts, product owners, and end users, allows requirements to be identified and understood. A software architecture defines a solution that will meet those requirements.

### ***Easing communication among stakeholders***

Software architecture and its documentation allow you to communicate the software architecture and explain it to others. It can form the basis for discussions related to aspects of the project, such as costs and duration. A software architecture is abstract enough that many stakeholders, with little or no guidance, should be able to reason about the software system

### ***Serves as training for team members***

The system's architecture and its documentation serve as training for the developers on the team. By learning the various structures and elements of the system, and how they are supposed to interact, they learn the proper way in which the functionality is to be implemented. A software development team may experience change, such as having new team members join or existing ones leave. The introduction and orientation of new members to a team often takes time. A well-thought-out architecture can make it easier for developers to transition to the team.

### ***Managing change***

Changes to a software system are inevitable. The catalyst for change can come from the market, new requirements, changes to business processes, technology advances, and bug fixes, among other things. Some view software architecture as inhibiting agility and would prefer to just let it emerge without up-front design. However, a good software architecture helps with both implementing and managing changes.

**Reorientation of SDLC w.r.t Architecture**

Ref 1

As already discussed in the above part of the research report, we have seen how an architecture is useful for implementation of software design. Now, in this section we’ll focus on how has the process of software development changed with the passage of time or how it evolved with time.

Let’s first discuss what are common elements among different models of software development.

**Commonalities of SDLC Models:**

* In every SDLC Model, there are some specific named activities to gather the needs of stakeholders. Apart from that, technical and specific requirements are also collected using different modeled activities. Those models which evolved later on also introduced business modeling and prototyping and building activities.
* In the old models there were limited system scope defining activities but as the model evolved, more activities were added. These included risks and win-win conditional analysis.
* In the old days, SDLC activities were limited to system requirements definition and specification. But now, SDLC activities have been improved with addition of system architecture, system requirements, definition, and architecture design.
* Every SDLC model has the same activities of some common types. They include analysis, design, coding, testing and implementation.
* Waterfall, Structured analysis, Unified process and MBASE include specifically the User manual in the SDLC process.

Above, we have discussed some commonalities of SDLC models. Now, let us move on to the distinctive features of SDLC models.

**Distinctive Features of SDLC Models:**

* Increment and Iteration: There are some SDLC models which repeat the same activities in different iterations. But some models repeatedly use distinct activities in the same way. For the first discussed model, example is Waterfall model while for the second one example is Spiral.
* Criteria for Iteration: There are some models which iterate the process when specific goals or objectives are achieved. On the other hand, there are some models, that simply repeat specific number of iterations. The difference is that, in the first case goals are set and measured that is green signal for the next phase. For the later discussed approach, only the increment in a phase indicates that goals have been met and process is moved to next phase of development.
* Complication: With the passage of time, newer model have been introduced with different analysis, design, coding and testing techniques that are more complicated and sophisticated that the older models. We may call it an advancement but in actuality it also increases the complexity and sophistication of the development process.

Reorientation / Introduction of unique elements in SLDC model

The table below describes the elements introduced in the different models of SDLC.

|  |  |
| --- | --- |
| Model | Unique Element |
| Waterfall | * Requirement * Analysis and Design Phase |
| SADT | * Strongly influenced by Systems theory and System engineering discipline |
| Prototyping | * Development of protype first instead of actual system |
| Structured cycle | * Quality control test * documentation control * user training |
| Spiral | * Risk analysis |
| RAD | * Scenario-Based Analysis |
| Win-Win Spiral | * Win-Win conditions added |
| MBASE | * Life cycle development plan * Phase plan * Product evolution plan |
| Component Based Cycles | * Component reutilization before building (a technology driven approach) |
| XP | * Agile Approach   (Stories to know environment and user needs, pair-based programmers and multi-role agents) |
| PSP | * Plan, Quality control records, postmortem analysis   (For individual software engineering) |
| TSP | * Launch and relaunch activities * Plan, Quality control records, postmortem analysis   (For teams) |

**Summary for reorientation of SDLC Models**

Ref 3

The key for all this re-engineering of SDLC models lies in the satisfaction of

* **User Requirements**
* **Budget Constraints**
* **Time Management**

The sole reason for the evolution of SDLC models was done in the chase of these goals. The process and activated were reoriented, modified, and changed with respect to better achievements of project goals. This evolution is still progressing because still there are projects that fails even if they follow recommended approaches and there are new horizons to explore for betterment of development activities and processes.

A major goal for all this research and evolution is to minimize the risk of project failure and maintenance costs.

SDLC models which are introduced earlier mostly focused on the activities that contributed towards the smooth process for development. From this we learned that, if processes are well defined and followed accordingly, time and budget can be managed well and hence this thing reduced the risk of project failure.

The model introduced later also took into account the process and culture of adopting changes in requirements. Not only this, but also the primary focus was to incorporate changes in both development and maintenance phase of development.

Future horizons need to expand exponentially, as the userbase will grow. There will be more influx of dynamic requirements and development processes will be in need of new models that adapt rapid growth and atomization of problems to be solved and managed easily. As we already know that the new process is the evolution of old SDLC models and crux of mistakes we learnt in the past, it is important to note that newer models must be adapted in the same fashion.

Ref 1

<https://www.researchgate.net/profile/Manuel-Mora-7/publication/232650502_A_DescriptiveComparative_Study_of_the_Evolution_of_Process_Models_of_Software_Development_Life_Cycles_PM-SDLCs_PDF/links/546e5ac90cf2bc99c215546c/A-Descriptive-Comparative-Study-of-the-Evolution-of-Process-Models-of-Software-Development-Life-Cycles-PM-SDLCs-PDF.pdf>

Ref 2

<http://www.cmnt.lv/upload-files/ns_85art05_CMNT2103_Magare.pdf>

Ref 3

<https://www.researchgate.net/profile/Pere-Tumbas/publication/267711880_A_Comparative_Overview_of_the_Evolution_of_Software_Development_Models/links/546310e90cf2c0c6aec1c6eb/A-Comparative-Overview-of-the-Evolution-of-Software-Development-Models.pdf>

**Conclusions**