**Faculty of Computing & Information Technology**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Title: SDA Research Report

Group Members:

|  |  |  |
| --- | --- | --- |
| **Sr #.** | **Roll Number** | **Student Name** |
| 1 | BSEF19M031 | Ata Ul Mohsin |
| 2 | BSEF19M033 | Hassan Ahmad Sarfraz |
| 3 | BSEF19M037 | Muhammad Saad |
| 4 | BSEF19M047 | Muhammad Saleh Butt |

Course Name: Software Design and Architecture

Date of Submission: Nov 12, 2022

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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**

SDLC mainly refers to a process that aims to design, develop and test a high quality software to meet the customers’ expectations efficiently. SDLC is not a methodology to develop a software but it is description of various phases involved in development of software. To develop an efficient software system, SDLC provides us with different methodologies like Waterfall Model, V-Model, Iterative, Agile, Rapid Application. Stages of SDLC are as follow:

1. Project Definition or Planning
2. Defining Requirements
3. Design the Architecture
4. Implementation or Development Phase
5. Testing the product
6. Deployment
7. Maintenance

**Project Definition or Planning**

This first and most fundamental phase of SDLC, the senior members of the team gathers information with inputs from customer, sales department, market surveys and domain experts in the industry. Basic understanding of the problem developed .In this phase, the basic approach to solve a problem and examine its scope is developed. Quality Assurance and risks posed to the software systems are also determined in this phase of SDLC.

**Defining Requirements**

Once we are done with the scope and understanding of the problem, the next phase is to gather requirements from customer or through analyzing market. Requirements can be mainly of two types

1. User Requirements
2. System Requirements

**User Requirements**

As the name implies, those requirements concerned with the end user are termed as the user requirements. These are the functionalities or jobs that user want to perform with a system with some input and system will in-return provides output to the user.

**System Requirements**

System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the customer's requirements. System requirements include the hardware specifications of the system to perform a certain task.

The requirements are written in a SRS (Software Requirement Specification) document that consists of all the requirements to be designed and developed during the project life cycle.

**Designing the Architecture**

The most important stage of SDLC, architecture of a software is developed on the basis of the SRS. The more clear requirements you have written in SRS, the best suitable architecture you are going to develop to meet the user requirements. Usually in practice more than one architectures are proposed based on SRS to build a system and most fit architecture is chosen. Existing architectures are also consulted.

The architecture is documented in a DDS which stands for Design Document Specification. The DDS is reviewed by stakeholders. DDS relies on various parameters such as risk assessment, product robustness, design modularity, budget and time constraints. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture are clearly defined in DDS.

**Implementation or Development Phase**

In this phase of SDLC, we are actually going to develop the product. The code is generated as referred by DDS during this stage. If architecture as proposed in DDS is not ambiguous, code can be generated without much hassle. Developers will follow any coding guidelines as defined by the organization and utilize different tools such as compilers, debuggers, and interpreters. The programming language is chosen with respect to the type of software being developed. This is longest phase of SDLC. During development, tasks are divided into modules or units and assigned to different team members.

**Testing the Product**

Once the implementation of system is done, it is deployed in the testing environment. We usually perform testing at every stage of SDLC and is referred as subset of all the phases of SDLC. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the quality standards mentioned in SRS.

**Deployment**

Once the bug free software is developed, the deployment of product starts. Based on feedback gathered from the stakeholders of the project, the final product is released and checked for deployment issues if any.

**Maintenance**

As we know, changes in software is inevitable, once the system is deployed, and customers start using the developed system, following three activities occur in general:

* **Bug** **fixing** – bugs are reported because of some scenarios which are not tested at all.
* **Upgrade** – Upgrading the application to the newer versions of the Software.
* **Enhancement** – Adding some new features into the existing software.

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

**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**

**Requirements:**

In the water fall model, first phase of activities was focused on what an application is supposed to do or developing requirements for an application. Requirements for an architecture are no the only thing we need to develop a system rather than they are a just a part of crucial components we need for developing a new system. That’s why, it is essential for us to have knowledge of previous architecture before moving on to the next one. In this regard, an example has been given as Greenfield development which is perhaps most dangerous. An architecture is the basis for a succeeding architecture.

**Design:**

In traditional models of software engineering, design activity is bind to design phase. Meanwhile in notational waterfall model, requirements are specified and after following design procedure a design is handed over to programmers for implementation. If some requirements are found to be inconsistent or infeasible, they are sent back to requirements phase for consideration to be examined again.

Additionally, a wide range of data regarding design techniques is required for an architect and he must deal with stakeholders issues, open source components and licensing issues. Matters like deployment issues, decentralized system concerns and upgradations are also to be handled by an architect.

**Implementation:**

With respect to implementation, reuse oriented engineering should be considered a good approach. For example, if we need to implement two functions from trigonometry, we may use an open source math library that may also include additional trigonometry functions. The important thing to note is that that architecture must be synchronized with the implementation of the system.

**Analysis and Testing:**

The fous

**Evolution and Maintenance:**

The fous

Mohsin 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**

Mohsin 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.

**Conclusions**

**References**

Mohsin 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>

Mohsin Ref 2 (Harvard formatted already below)

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

Magare, A. and Lamin, M. (n.d.). *Cognitive evolution in software development life cycle through design thinking*. [online] Available at: http://www.cmnt.lv/upload-files/ns\_85art05\_CMNT2103\_Magare.pdf.

Mohsin 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>