Evolution of Software Systems from Incubation to Enterprise Solutions

Manimegalai R[1], Vishnu Charan[2], and Venkateshwaran M[3]

{[1]drrm@psgitech.ac.in, [2]vishnucharanoss@gmail.com, [3]19cs156@psgitech.ac.in}

[1,2,3]Department of Computer Science and Engineering
PSG Institute of Technology and Applied Research
Neelambur, Coimbatore, India

Abstract. The review paper includes the milestones and histories involoved in Software Engineering and the usage of various software procedures and methodologies in respective timeline of software development. Software architecture plays a major role in company development. A company should start from single tier architecture at first as it aids in easier development and advancement but it has its downsides too. A single point failure or an error in single tier system may lead to serious damages and loss. Hence it is mandatory to start development from single tier architecture and migrate to multi tier architecture in future. Multi tier architecture invloves distributed computing and cloud technology to avoid traffic and single points of failure. Introduction of microservies in multi tier architecture opened way to standalone applications and easier feature enhancements. Once development is done, deployment is made easier by the introduction of Continuous Integration and Continuous Deployment with network of nodes. Logging and auditing should be made mandatory to rectify problems occur during development.

Keywords: Software Engineering, Software Systems, Single Tier Architecture, Microservices, Multi Tier Architecture.

1 Introduction

In today's world, where creation of software applications and products have never been easier, software enterprises and startups have been steadily on the rise. However, out of several startups that come into the market, only a handful succeed in fulfilling their expectations and establish a solid stand in the technology market. This is because most software startups lack a long term strategy. Every startup gets

ICSETPSD 2023, November 17-18, Coimbatore, India
Copyright © 2024 EAI
DOI 10.4108/eai.17-11-2023.2342836

initialized with a low investment, due to this reason, most of the startups always focus only on their short term goals and meet the required prospects. However, without a proper plan even though the startups succeed in the initial phase, they are sure to experience a business failure in the near future. This is why software startups work on analyzing their field of business, collect all the required data for the working and testing of their applications. The final end product which is the software application, needs a well devised architecture to successfully execute its business logic.

In terms of architecture, it is always advisable for the startup to initially work on the Single tier system architecture and once the traffic becomes normal it can be converted into Multi tier system Architecture. Multi tier system architecture and its success predominantly depends on the cloud technology[3] and the efficient usage of virtual machines. Generally these Multi tier systems run on different virtual machines but on the same server. This way each system has its own allocated space. However its comparatively difficult to code Multi tier architecture compared to Single tier architecture.

Netflix is a good example of company that successfully converted its application and business logic from the Single tier architecture to Multi tier system architecture. Programmer obtains a strong foundation in terms of business logic when he understands the code and moves his application from the Single tier architecture to multi tier architecture.Selection of a proper architecture is the root for a successful business, once that gets done it is a much devised and planned procedure to proceed on working the application as it hits the market.

2 Milestones and Histories in Software Engineering

The SE timeline's and periods are described in Table 1 along with the readily available and most used Procedures. We begin from 1956 since it is commonly accepted that General Motors initiated and created the inuagral and very first operating system in that year.

Table 1.
Software engineering timeline.

Software Engineering[12]			
Timeline	Terms	Description	Procedures
Mastering machine (1956–1967)	Batch	Hardware dependent high level languages	
	Interactive	Online. Code and fix	

Mastering process (1968–1982)	Process	Crisis. Development process software engineering	SREM, SADT DSED, JSP SSADM
	Formal	Ensure correctness. Models inapplicability in big problems	
Mastering complexity (1983–1992)	Structured	Personal computer. Expanding data and functional convergence	Modern SSADM JSD OMT Booch Jacobson
	Object oriented	Reusing new programming approach	
Mastering communications (1993–2001)	Industrial	Internet. Client/server complex projects	CORBA RUP/UML
	Distributed	Integrated methods quality	
Mastering productivity (2002–2010)	Abstraction	Conceptual level expansion Customer productivity Customer involvement	MDA XP Scrum
	Agile		
Mastering market (2011–…)	Service	Outsourcing services Orchestrating services Market demands. Downloads	BPMN/BPEL SOA-Cloud APP
	Mobility		

During 1956, Software engineering was not yet a recognized phrase. Outside pressures had a significant influence on code development. Any piece of software's primary goal was to maximize the use of the constrained hardware resources. This led to high risk which paved way to the birth and coinage of Software Engineering[2]. Advances in data and function modeling conisisting database and structured methods mastered the complexity with the inclusion of object-oriented methodologies. The introduction of internet, concurrent and distributed programming, and various maturity models mastered the communications. The introduction of various SE procedures and methodlogies mastered the productivity. The introduction of N-tier, Service Oriented and Multi tier architecture keeps on mastering and advancing the market.

3 Methodologies and Procedures in Software Engineering

Methodologies and Procedures that are at present in use are an evolution or a fusion of those that were developed and used in earlier periods. The distinctive characteristics of a few different SE development approaches are compiled in a succinct description in this section. Structured methods[9], object-oriented approaches, and agile processes are the three categories of techniques that we distinguish. The most pertinent approaches are compiled in Table 2 along with the key characteristics of structured methodologies, some of which are no longer in use but have a significant impact on current methodologies.

Table 2.
SE methodologies summary.

Artifacts	Notation	Plans
Structured system analysis and design methodology (SSADM)[12]		
Requirement specification Analysis model Design model	Data flow diagrams Data dictionary Structured English Structure chart	Specification or analysis Design Coding and test Maintenance
Data structured systems development methodology (DSSD)[12]		
Data model Functions Results	Data structured diagram Warnier/Orr diagram Assembly-line diagram Entities diagram	Context definition Function definition Results definition
Jackson system development (JSD)[12]		
Initial model Functional model	Entity life history diagrams Structured English	Entity/action step Initial model step Interactive function step Information function step System timing step System implementation step
OMT methodology[12]		
Object model Dynamic model Functional model	Class and object diagram Modules diagram States diagram Process diagram Interaction diagram	Conceptualization Analysis Design Evolution
UML and RUP[12]		
	Class diagram Use case diagram	Dynamic: inception, elaboration,

Use case model Analysis model Design model Deployment model Implementation model Test model	Interaction diagram State diagram Components diagram Activity diagram Components diagram Deployment diagram	construction, and transition Static: business modeling requirement, analysis and design, implementation, test, and deployment
	Extreme programming[12]	
Software releases All SE techniques	Communication Feedback Simplicity Courage Respect	Coding Testing Listening Designing
	Scrum[12]	
Software releases Meetings	Main roles: Scrum Master Product Owner Team	Sprint planning meeting Daily Scrum meeting Team work Sprint review meeting Sprint retrospective

Procedures and Methodologies like SADT and SREM combined Structured System Analysis and Structured System Design (SSADM) introduced in structured programming which enhanced modularization extension and extended information hiding when migrating from design to specification phase. In nineties, the concept of objects made way to object-oriented procedures and methodlogies to replace structured programming which was used in the seventies. On using these methodologies had a downside conflicts on similar methods. So they initiated the Unified Modeling Language (UML) design. These collaboration between designing and methodologies gave interoperability. Many recent advances in software engineering gave a clean software management methodology called Agile. In this method, the entire project is divided into timeboxes and each timeboxes are assigned with required resources(stakeholders). Timeboxes are usually small pieces of a large project

4 Formation and Assembly for Startups

In today's world every software application has its roots starting from incubations and startups. Essential communication at the early stages pose the foundation, laying the blueprint for the entire software application. Typically in a startup the employee count generally range from an average of 200 with 40 residing in multiple departments. Passing of valuable information and total transparency between departments will serve as an important pillar for successful production of the application prior to the deadline. For all this to happen employees must have a basic knowledge of the entire skeletal picture of the application. This can be made possible only in a single tier software system.

Single tier software systems are a packed cluster that contain both frontend and backend information. The reason that the single tier software systems are more beneficial in the initial stages[1] is because it is much easier to pass on information from one team to another as all of them share the same root project. Any essential change or version update being done would notify the entire team of the new changes, this would streamline the most crucial advancements for the application. Also in the initial stages, it's important that the project always stays under constant supervision. Traffic for a software application[4] refers to the User count currently being active and using that software system, essentially for these software applications the traffic would be considerably low during the initial stages. Therefore these single tier systems work well in this scenario as it would take up less space and time complexities if the traffic is low, therefore the entire application can be deployed without any hassle. This could also cut down cost to a very high level for the company.

Once the application is deployed into the server the first time, it is much easier to manage and update too. And before sending the application into the market, all the modules of the application go through a phase of software testing, where the business logic, flow and the performance is being tested. To state with an example, if a software application has a login module for the user, profile module for user information and user history module as well, all these modules are interlinked as all these three modules share the same user data, therefore during the testing phase it becomes easier to test these three modules if all three exist under the same repository. Since the results of one modules may depend on the other, if the logic for all the required modules exist together it would be much easier to test.

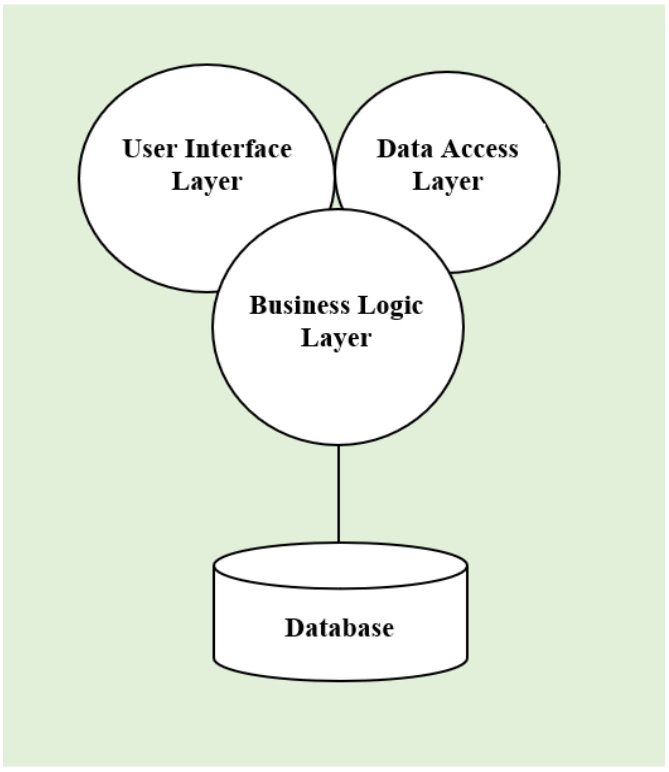

Fig. 1. Monolithic Architecture

5 Downsides of Single Tier Systems

As the single tier software systems prove to be a great choice, they do come with their own drawbacks as well. The application is prone to several security risks if the entire business logic exists under one common root project[7]. Generally a typical user does not visit every page in a web application, as every user has their own needs and they utilise the software system to just complete their request, it would be completely useless to provide the user with every page. But this could be a problem in terms of Single tier systems as they do not have any other choice except to provide everything because both frontend and backend[11] for all the pages are packed together under common deployment.

The front end and backend information are volatile as they undergo changes dynamically for each user, however the data that resides in the database needs to be static and undergoes changes only when prompted. So when the web application is under attack, if the database does not have any replica, the user data and any other crucial information might get lost, therefore its not always a good choice to pack all the information together.

6 Multi Tier Software System

A multi tier software system generally refers to the microservice architecture. A single tier system typically contains all the code and the required data stacked up in one place but in the case of a multi tier system all the codes are separated into individual deployable chunks that are interdependent. With the advancements in the cloud technology, multi tier system architecture has reached great heights. Earlier in single tier application the entire code logic exists in one main project. Therefore for any new update the entire application needs to undergo change, rebuild and needs redeployment. However in the case of multi tier system, since the main business logic is separated into multiple chunks of code. If a particular service needs an update or change it does not affect the other services, only that particular service needs to undergo rebuild and redeployment[6]. This could save a lot of time.

In the initial stages due to lower traffic and demand it is preferable for the startups to program their software system using the Single tier architecture. However as their software system widens and the traffic increases, it's a better choice for the startups to adapt to the multi tier architecture. As this is highly scalable and helps the startups to meet up with their demands and deadlines much easily.Also since in the initial stages for a startup, they lack resources and a strong workforce, it is important for them to easily detect and understand the flow of code. This is difficult in the case of a Single tier architecture as the codebase and logic is quite vast. But when that codebase is broken down into smaller chunks it is much easier for the team to understand.

To explain in terms of a general scenario, suppose a customer visits the website of a National Bank. This website generally comprises of several services offered by it. However the user does not visit all those services, he simply wishes to just satisfy his need and completes it. When this is imagined in terms of Single tier architecture, the server loads the entire application in which only a certain part of it is being used. However in terms of Multi tier architecture, since each service is deployed independently, when the user makes use of a service he does not need other services, hence this saves up much memory and therefore increases the performance. However every architecture has its own downsides, in a single tier architecture, since one common codebase contains the entire business logic, it can be easily deployed in a set of servers. However for Multi tier based architecture it is deployed among several servers, therefore each of these multiple services being deployed needs to be properly monitored.

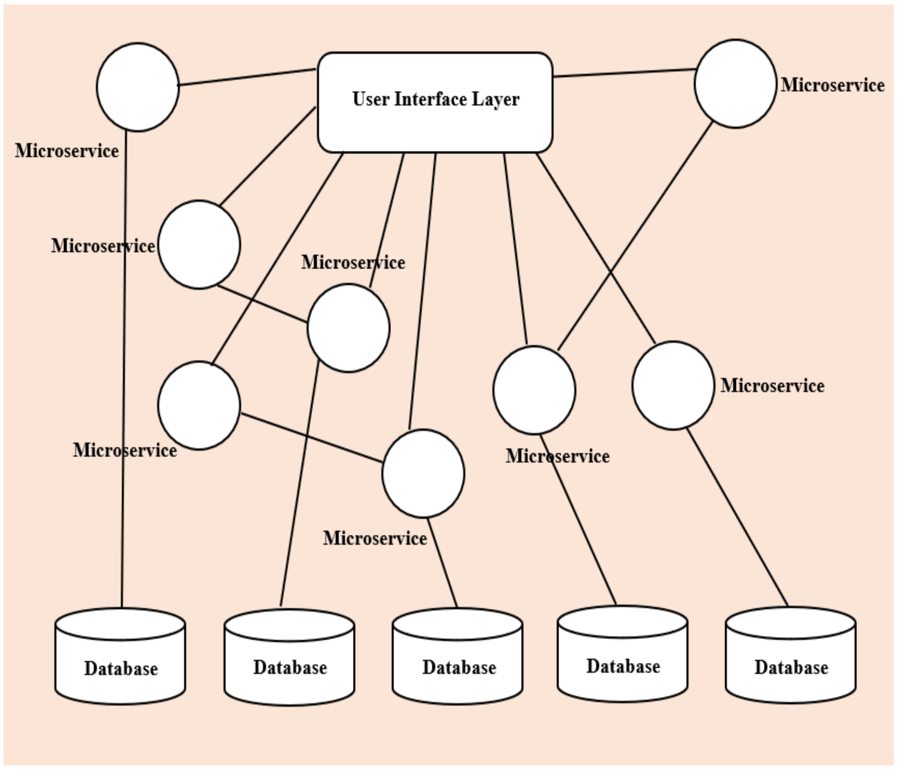

Fig. 2. Microservice Architecture

And increased coordination from the team side is required while working on multi tier architecture, this is because while working on the single tier architecture the entire code resides in only one codebase therefore it is easier for the programmer to understand not just his code but also someone else's service which might be needed by him, but in terms of multi tier architecture since each programmer has idea on his service alone he needs to have better clarity on other independent services as well for proper integration[8] and communication between multiple services. This causes issues in terms of global testing, since communication between two different services might be needed sometimes, testing that dependancy in service communication can be difficult. In a startup level this can be highly difficult as team chemistry is still budding.

It is always a general practice for teams to being with single tier architecture and then shift into multi tier architecture which migrates into microservices[5], this enables the team to have a broader picture in terms of business logic while programming for the single tier architecture as it contains the bigger picture, however after the codebase is established tested and deployed in the market, it is a more viable option to shift to the multi tier architecture style which is cost efficient.

And by this time team collaboration would also have reached stronger foundations therefore it would be easier for the teams to shift their code from the single tier architecture to the multi tier architecture.

7 Seamless Collaboration

To meet deadlines, boost up productivity and provide regular updates, it is highly mandatory that automated integration of code changes made by multiple people comes into picture. This is done with the help of Continuous Integration (CI). With the help of CI people belonging within the same team or multiple teams can work on their changes and collaborate their changes into a single final product. Consider that the main project exists in one space, and each person or team works on adding new features to that main project, so from the current state of the project, they build on their changes within their own allotted space. Once they complete their changes, proper evaluation and reviewing takes place, then the changes can be included within the space of the main project, and that becomes the main state of the project.

This way multiple updates and add-ons are done to the main project. Without CI it would be a great problem, if every person works on the main project, it leads to issues like code overwriting, removal of essential code or data, version irregularity etc. This CI pipeline provides features to both run the code as well as test the code. Since CI provides user friendly code monitoring capacity, the team lead can approve the code only if the code meets the necessary test requirements.CI tool is the most important concept in the area of Developer Operations (DevOps).

8 Automated Deployment

The final outcome of any software system is successful deployment to a server, as it can be used not just locally but on a network or a cluster of nodes. This can be done with the help of Continuous Deployment (CD). A CD pipeline[10] takes care of the deployment of any software application. Startups must adapt the idea to deploy their application in spaces. These spaces denote the testing levels of the software application. Each space undergoes rigorous testing under multiple levels so that a refined product can be squeezed out in the end. For this to happen the deployment of the software application must take place in all these multiple spaces progressively. This is done with the help of Continuous Deployment Pipeline.

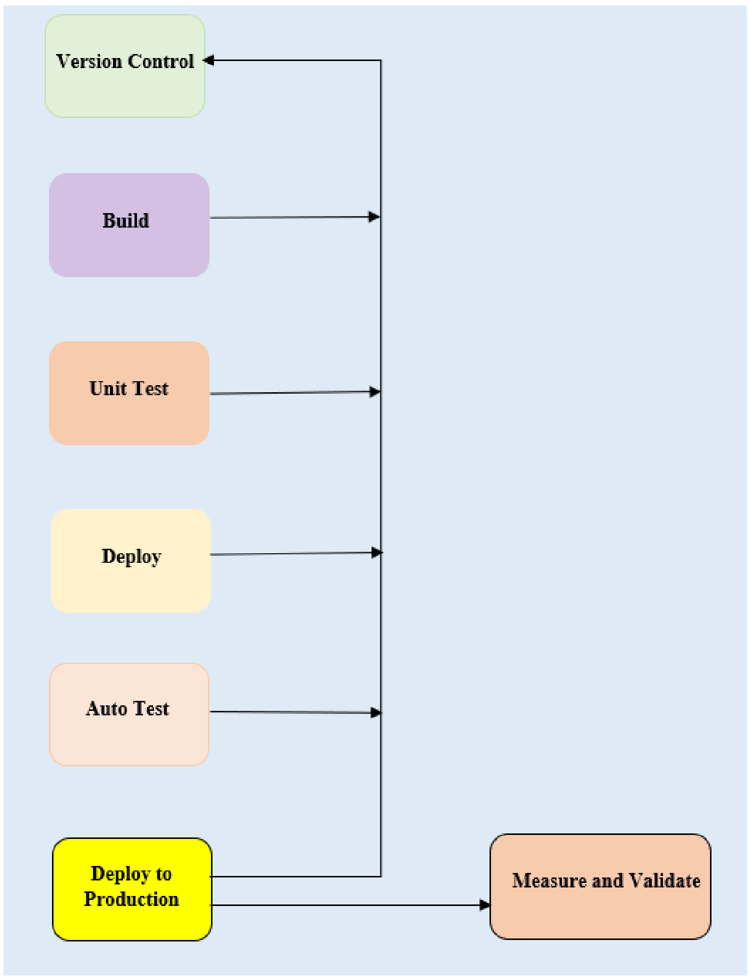

Fig. 3. CICD Pipeline

9 Conclusion

Software systems are dynamically handled through multiple teams over a period of time. Different people work on a particular code segment to provide updates, and since teams always work on a constrained time frame to deliver the code prior to the deadline, it's highly important that it becomes easy for each person to debug a code or correct any errors. For this to happen proper logging and auditing of the program scripts needs to take place. Logging at appropriate places gives the programmer an idea about the flow of the program. Therefore even if the same script gets handled by multiple programmers over a period of time, it becomes easy for them to understand the code and find the errors easily. So in the initial phase it is highly mandatory that a programmer properly includes logs for his code so that it can be made easy for others to understand the flow of code. Logging in the means of documentation aids for easier auditing.

References

[1] Blinowski G., Ojdowska A., Przybyłek A.: Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation// IEEE Access -Vol. 10, (2022), s.20357-20374

[2] F. Meziane and S. Vadera, *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, Advances in Intelligent Information Technologies, Information Science Reference, 1st edition, 2009.

[3] Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In O. Gonzalez, & M. Sanchez (Eds.), *2015 10th Computing Colombian Conference (10CCC) : Universidad de los Andes, Bogotá, Colombia, September 21st to 25th, 2015* (pp. 583-590). Article 7333476 Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/ColumbianCC.2015.7333476

[4] M. Harman, "The current state and future of search based software engineering," in *Proceedings of the Future of Software Engineering (FoSE '07)*, pp. 342–357, IEEE Computer Society, May 2007.

[5] L. Händel, 'Microservices in the context of a fast-growing company', Dissertation, 2020.

[6] Al-Debagy, Omar and Peter Martinek. "A Comparative Review of Microservices and Monolithic Architectures." 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI) (2018): 000149-000154.

[7] Arachchi, S.A.I.B.S. and Indika Perera. "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management." 2018 Moratuwa Engineering Research Conference (MERCon) (2018): 156-161.

[8] Shahin, Mojtaba et al. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices." IEEE Access 5 (2017): 3909-3943.

[9] B. G. Buchanan, D. Barstow, R. Bechtal et al., "Constructing an expert system," in *BuildIng Expert Systems*, F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, Eds., pp. 127–167, Addison-Wesley, London, UK, 1983.

[10] Chaudhary, Ashutosh & Gabriel, Mary & Sethia, Rishabh & Kant, Shubham & Chhabra, Sonia. (2021). Cloud DevOps CI -CD Pipeline.

[11] Auer, Florian & Lenarduzzi, Valentina & Felderer, Michael & Taibi, Davide. (2021). From monolithic systems to Microservices: An assessment framework. Information and Software Technology. 137. 106600. 10.1016/j.infsof.2021.106600.

[12] Águila, Isabel & Palma, Jose & Túnez, Samuel. (2014). Milestones in Software Engineering and Knowledge Engineering History: A Comparative Review. TheScientificWorldJournal. 2014. 692510. 10.1155/2014/692510.