A Review on Processes and Tools used for Secure Software Development and Their Importance

Ekanayaka E.M.S.G.
Faculty of Computing,
Sri Lanka Institute of Information Technology (SLIIT),
IT19121048

Chandrasena T.K.T.A.
Faculty of Computing,
Sri Lanka Institute of Information Technology (SLIIT),
IT19099514

Nandana G.M.D.
Faculty of Computing,
Sri Lanka Institute of Information Technology (SLIIT),
IT19104690

Sithpavan. G
Faculty of Computing,
Sri Lanka Institute of Information Technology (SLIIT),
IT19134468

***Abstract*— Because of the rapid progress of technology, the world has become an integrated platform with a wide range of processes and systems that have constantly developed mechanical and interrelated sub-systems. A proper software development process (SDP) is now required to create that platform. The software development process is considered the root of those interconnected subsystems, so those processes exist as a significant driving force in mechatronics with the design, development, and maintenance of smart systems. This review paper focuses on what a software development process entails and how to create secure software that is suited for both manufacturers and relevant key parties. When it comes to manufacturers, it refers to the developers, software engineers, and other engineers involved in the development. With the availability of software devices and cloud services, the need for connecting the device/software to the internet expands, as does the requirement for secure software development (SSD) process for software developers. As a result, SSD must be transformed into a tool on which developers can rely for secure software as well as the ability to withstand cyberattacks. This review paper also discusses previous research papers on the content and implementations of secure software developments, as well as how they are applied to work developments. Aside from that, the definition of a secure development life cycle (SDLC) covers flaws of secure software coding practices, as well as the issues associated with them. Following that, methodologies for improving the software development process's security standards are proposed to resolve security vulnerabilities. Finally, the study concluded that the feasibility of incorporating security features into software development phases to create secure is dependable on application programs and systems.**

***Keywords*—**

I. INTRODUCTION

When selling software to a customer, you must protect your centerpiece by establishing trust in your software while maintaining the flexibility and speed that will keep you competitive in your market. Many organizations use security measurements with the software development life cycle (SLDC) to secure the centerpiece [1]. However, some organizations are currently lacking in terms of building those [1]. This is because, after implementing the application, developers must change the codebase and components for various security reasons, causing a problem that forces developers to rework on the project. This also causes a delay in launching new features to the market. Businesses are exposed to risk when using insecure software. When new features are introduced, security methodologies must be followed because failing to do so exposes application privileges to hackers. The development team must incorporate security by designing secure software processes that facilitate, rather than impede, the creation of high-quality, high-security products for your business. With that secure software development becomes a priority when developing an application.

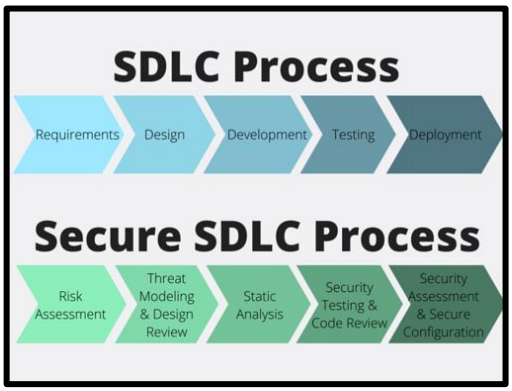

Figure 1: SLDC process vs Secure SLDC Process

A software manufacturing method called secure software development (SLDC) emphasizes security throughout SLDC [2]. The four simultaneous activities, as well as the four phases, make up the bulk of the software development process. These phases are requirements, implementation, design, and evaluation. Forward engineering, safety, reliability analysis, and project management are among the operations mentioned above. Earlier, software was installed on CDs, and no new features or updates were introduced. When installing a new version of that application, a new CD must be used. When new software versions are directly introduced, they have an impact on the application's security vulnerabilities. Since the user regularly uses the application, they encounter different procedures to hack the application [3]. This is due to lack of security updates. When comparing applications running on wired and wireless connections, wired connections have a lower spread of security vulnerabilities because wired connections do not require internet connections [3].

To integrate security with software development pattern, security must be taken into account with the beginning of the SDLC. [4]. The practice of designing, creating, and testing software to ensure its security is known as secure software engineering (SSE) [5]. Both secure SDLC procedures and secure software development (SSD)

techniques are included. Security is typically viewed as an afterthought by enterprises [6]. At some point during the pre-development stage, security is not taken into account [7]. A small mistake can cost millions of dollars in today's corporate world. Unfortunately, many software development firms do not follow the recommended SDLC incorporation best practices [8]. The Internet of Things (IoT) and the Internet of Every Thing, as well as the development of Internet-based software systems, cloud computing, social networking, and location-based services, as well as new business paradigms, a variety of customer requirements, the rapid advancement of ICTs, and new regulations, all contribute to the rapid evolution of software applications. [2]. Software quality is increasingly impacted by security concerns as software development grows more intricate, distributed, and multi-faceted [10]. In addition to damaging an organization's reputation with clients, business partners, and investors, non-secure software also raises costs by enabling employers to rebuild unreliable applications and impedes other future development by devoting limited resources to fixing current software flaws [10]. Most software applications are developed and implemented with minimal thought given to security [11]. Daily hidden assault risks that compromise the accessibility and integrity of organizational data result in significant financial loss, loss of secrecy, and loss of confidence [12].

It is logical to make sure that a program or software is not only simple for users to use and understand, but also that the program itself is as secure as possible and capable of preventing potential hacks or attacks that could jeopardize the entire program. This is especially true when the program or software is being developed for use by a large group of people within an organization. Therefore, in order to reduce the dangers indicated, it is essential to establish a software system implementation with security in mind during the development process [3].

The sad rise in software security breaches in recent years is partly attributable to developers' attitudes, who think that software security should only be taken into account after the development phase of the Software Development Life Cycle (SDLC) [14]. If security had been thoughtfully planned throughout the development process, as opposed to being implemented at the last minute or at the conclusion, this issue might have been easily avoided. Every layer of software must be as secure as it can be from flaws that particular software might experience. There is another reason for an increase in security threats in addition to the previously indicated rationale for unsafe applications [3]. The Internet of Things, wireless networks, and other technologies are developing in this way (IoT). Although it may seem like these technologies have made life simpler, they have also increased insecurity in terms of data transmission, information reliability, and susceptibility to physical and cyber-attacks [15]. For instance, there have been more data breaches and network and routing intrusions [16]. Therefore, it is crucial to include security features at every stage of the software development process and to maintain and upgrade the security components on a regular basis. The following section examines the software development life cycle in this regard.

II. RELATED WORKS

In this paper [17], They provide a software development process for creating secure software in their study [17], which consists of four fundamental tasks. These include program management, reliability analysis, safety analysis, and forward engineering. Cardiac Pacemakers Inc. CPI develops cardiac rhythm control devices that are biocompatible [17]. CPI has developed a methodology to software development that emphasizes the importance of developing secure systems and products. Because software safety can be discovered in relation to system obstacles, CPI defines the procedure in the perspective of a system lifecycle. CPI is aware that while safety has largely been ignored in the industry, software quality has been a topic of concern for quite some time [17]. Four separate phases and four parallel activities make up this division of the software development process [17]. Requirements, design, implementation, and evaluation are the phases. The phases are requirements, design, implementation, and evaluation. Among the tasks are forward engineering, safety analysis, reliability analysis, and project management. System maintenance and defect evaluation and monitoring are three important support procedures included in the SDP. [17]. The incorporation of hazard analysis, which constitutes the primary focus of this study, even though the general stages of this process are not. Only the information necessary to comprehend how other activities are assimilated into or influence of the safety evaluation will be covered.

This paper [18] mentions a case-based control system for safe software development that combines an artifact & knowledge-management systems. Systems for managing artifacts and secure software lifecycle information are known as artifact management and knowledge management systems, respectively. Furthermore, the connection between artifacts and cybersecurity knowledge is managed, with the support of visualization of software security knowledge and artifacts and software security knowledge. [18]. They also ran an experiment to verify the value of knowledge retrieval through knowledge base visualization. In addition, even though cases are applied in many languages, the implementation phase of countermeasures varies, necessitating the use of cases that must be applied in various tongues.

In this study [19], the researchers sought to construct a framework for simulating the adoption of security practices in software development and looked into possible sanctions for developers who embrace security practices more frequently. A multiagent simulation framework with developer and manager roles is suggested [19]. In this framework, developers are responsible for

maximizing task completion and compliancy with security policies, and the manager is responsible for enforcing sanctions based on the functionality and security of the project. They use a case study from the actual world to illustrate the model using a 13-year database of bug reports for the Eclipse Java Development Tools to initialize the model's incidence of errors. The findings demonstrate how strongly developer preferences affect security procedures. They found that consistent punishments could result in decreased developer retention and a general decline in security standards. The model contrasts security tool adoption among engineers with different preferences and offers managers advice on choosing the best sanctions for promoting the use of security tools in software development.

According to the [5] research paper, SDLC should give functionality, flawless quality, and security measures in every stage. Despite this, the aim of the software developer is to deliver all the functionality in the shortest time possible. A security vulnerability that has caused a casualty may take additional time, money, and resources to fix. Security controls must be incorporated into the SDLC in order to create secure software applications. The developing procedure takes more time and does not provide complete attack resistance. However, using the methodology reduces the likelihood of an attack. quality and secure software application in the end. As a result, the public and peer groups will contribute to the company's reputation.

The paper [21] discusses traditional vulnerability research approaches, which are less effective and adaptable and frequently require security researchers to have specialized training and a wealth of real-world experience. Deep learning and natural language processing technology can currently be used to support security vulnerability research, which also improves security vulnerability mining. The binary vulnerability detection method has broad applicability and good detection accuracy, but it is difficult to find out the structure and type of the higher-level code of the program. Disassembly operations are now employed to convert binary code into assembly instruction codes to find vulnerabilities in binary code. We can employ program analysis technologies to extract important vulnerability code information from these programs, vectorize it, and then use it as training material for neural networks. The static analysis technology further divides the vulnerability processing technique into two groups: a vulnerability detection strategy based on code similarity and a vulnerability detection technique based on code pattern.

In the research paper [6] discuss Agile software development security. According to the paper, the emphasis on preventing latent security issues necessitates that software security engineering begins in the very beginning of the software life cycle. This offers the chance to first stop security incidents from happening. Additionally, it can enhance any external security measures put in place during the life cycle of an application's operations and maintenance. Security problems, such as implementation errors and design defects, frequently exhibit persistence. If left unfixed, they develop into "features," necessitating ongoing and expensive security engineering work for the duration of the software's operational existence. Early fault detection and repair uses a lot less resources than fixing problems after the fact. Furthermore, difficulties with users, hardware, and other aspects of the operational environment are more sporadic and are less dependent on the assets protected to be mitigated. Software security engineering differs from other security disciplines due to these features.

In this paper [7] explains the creation of an intervention that improves security, how it was used in 8 different organizations, and its theoretical and practical implications. The investigation searched for and measured advancements in "assurance techniques": process enhancements, knowledge gains, and skill enhancements that would result in improved security over the long run. Apart from this contribute to how the security of team-developed software can be improved by a straightforward "intervention package" composed of a series of workshops that are facilitated an understanding of the significance of and developers' abilities to communicate security improvements in terms of their business benefits.

According to the latest research and studies [8], Cybersecurity specialists must find ways to include cyber security with software development processes due to practices like open-source development, agile, DevOps, and DevSecOps. Many organizations are working on teaching software developers about cyber security as one method to handle this issue one of awareness, education, and training. Security should be "moved left" in the software development process, according to cyber security experts, so that it is taken into account early on in a project. Cyber security is a difficult problem in and of itself, impacted by both social and technological aspects. In programs to influence behaviour, Social Practice Theory (SPT) is being employed in areas as diverse as human consumption and environmental challenges. SPT is the study of a person's preferences, routines, and behaviours that emerge in a social setting, but instead of using an individual as the analytical unit, it looks at 'practices'. In order to increase security within a software development lifecycle, it is necessary to consider both social and technological factors. Culture has a significant impact in software development processes.

In [9], the authors discussed how security has been found to suffer as a result of large-scale agile methodologies. Although the features and application domains of these approaches vary, they all share the fundamental ideas outlined in the Agile Manifesto. Agile approaches, according to previous studies, enhance knowledge sharing, communication, team productivity, and product quality. Other research, however, has indicated a detrimental effect of agile approaches on security. The authors made a distinction between the three types of security difficulties that have been identified: general security challenges, agile-specific security challenges, and large-scale agile-specific security challenges. An example of general security challenge is it has been observed that a problem common to all software engineering projects is project management sacrificing on security owing to a lack of resources. The research discussed the differences

across teams and roles and how it affects the security in large-scale agile projects. The level of security knowledge is a topic of disagreement among several teams. The obstacles are seen differently by respondents in various agile jobs. Software engineers and system owners faced security concerns that were more practical in nature. Product Owners, a role more directly associated with business operations than with IT, faced challenges that were more administrative in nature.

As discussed in this article [3], the operational tactics of teams are frequently neglected by best practices. One instance is the division of labor, where adhering to best practices would need certain teams to restructure and redistribute tasks—an effort that is often considered unrealistic. Company culture, security expertise, outside pressure, and having experienced a security event are further determining variables. The extent of security integration in the SDLC was once thought to be strongly correlated with the size of the firm. Their research indicates that it may not actually be a deciding factor. In their dataset, the team from the smallest firm actually outperforms the team from the largest company in terms of security attentiveness. Furthermore, they found no proof that development techniques had an impact on security procedures. Even while our information prevents us from drawing any firm conclusions, it reveals a worrying pattern of low-security adoption in many of our project teams.

In this work [10], the authors discussed that security flaws in source code are the root cause of a lot of the security issues that individuals today encounter, including security breaches and data theft. Their research showed that software developers would first complete their implementation duties, regardless of whether the firm used an AGILE approach or a more conventional waterfall process and SDLC (Systems Development Lifecycle). Their auditors did remark that developers often performed code reviews and testing to check for a variety of defects and quality problems, but that the auditors were excluded from that process as security was rarely a priority for them. All of their research participants had the primary duty of identifying security flaws in the source code of the applications. As anticipated, every participant mentioned utilizing Fortify2 and AppScan3 as well as static and dynamic analysis tools to find security flaws. All static and dynamic analyses were carried out independently by the participants. Within an organization, static and dynamic analysis were occasionally seen differently and completed at various periods. To list the prevalent security procedures mentioned by their participants: finding security flaws in code is the primary responsibility of security auditors, their procedures need a lot of manual effort, and development teams are in charge of patching vulnerabilities.

This paper [11] contrasts a well-liked secure software approach with a fresh, flexible secure software methodology. The organization in charge of this experiment evaluates a real-world experiment using information from actual software projects as a case study. Two development scenarios were used to summarize and compare the results: traditional, with a reactive security strategy, and emerging, with preventative measures and security by default included throughout the software life cycle. The case study's overall number of vulnerabilities is decreased by 68,42%, which also lowers their criticality and the amount of time needed to fix them. The presented approach shows that the more recent solution offers better secure software by methodologically enhancing software security and quality. The paper claims that they primarily focus on four key areas. An actual software project that can validate the idea, safe software development models that have undergone in-depth analysis and comparison, and secure software processes enhanced via the deployment of the suggested model is among them [11].

The primary goal of this paper [12] is to explore security measures in the setting of creating secure software while conducting systematic mapping research (SMS). Based on the specific criteria, 116 research studies were selected. The 116 papers were classified after gathering information by quality evaluation, security software techniques, SDLC transition periods, and SWOT analysis. The findings suggest that this field continues to be in its infancy and that more research, particularly on solutions that have undergone empirical evaluation, is required.,

The paper's [13] objective is to develop a framework for enhancing the secure product creation procedure in software firms. A Multivocal Literature Review (MLR) was carried out to find pertinent studies in both formal and grey literature in order to achieve this goal. To construct a Secure Software Design Maturity Model, a total of 38 main studies were found. Then, the relevant information was compiled into 8 knowledge categories and 65 best practices (SSDM). The framework was developed utilizing the CMMI v2.0 structure and put to the test in actual scenarios using case studies. The results of this case study showed how SSDMM may be used to assess an organization's readiness for the secure design phase of the SDLC. Organizations can evaluate and enhance their software design security procedures with the use of SSDMM. Additionally, it will set the foundation for new software security methods to be developed by researchers.

This paper [14] presents a model that uses fuzzy multi-attribute decision-making (FMADM) and adds group decision-making. Both survey ranking and case study evaluation employ the FMADM methodology. With a collection of 101 publications, a systematic literature review (SLR) is first written to identify all referenced difficulties. They have identified 18 problems that GSD vendors run across when working with OSDOs. Six were deemed to be absolutely necessary. In order to find corrective interventions, a second SLR was carried out, and 75 corrective measures were taken from 63 chosen articles. To substantiate our SLR findings, we surveyed 42 outsourcing specialists from six different nations. In addition, based on CMMI, SOVRM, and SOPM, six significant issues and 75 corrective practices were divided into four mitigation levels. Additionally, two case studies were conducted to evaluate the results of CCCMM in OSDO companies. Company A remains at level 1 since the assessment findings from the first case study do not suggest that Company A implement

CCCMM level-2 successfully [14]. According to the second case study, Company-B has only successfully completed level 2 of the suggested evaluation model after implementing all of level 2's essential problems. They identified 17 communication and coordination issues that OSDO vendor groups confront through SLR1. Six of these problems were thought to be critical. Using SLR2, we identified 75 practices for these pressing issues, and an OSDO industry questionnaire survey served to confirm them. As a result of this research, a framework model (CCCMM) based on CMMI and the FMADM technique is being created to project the likelihood that an OSDO connection would be effective [14].

III. Methodology

- **Secure Development Life Cycle (SDLC)**

The secure software development life cycle (SSDL) includes sets of standards that support security assurance and compliance requirements [31]. By following these methods, developers can create more secure software systems with low development costs and a low no of vulnerabilities. These SDL standers can be applied to other variety of software development life cycles (SDLC) like a waterfall, Scrum, Agile etc. for every SDLC there are common phases, Planning, Design, Testing, Deployment, and Maintenance.

Figure 2: SLDC Cycle

Frequently during the testing phase of the SDLC, most of the security vulnerabilities of the developed system can be found [32]. This is not a good practice when developing a system because in the Planning phase of the SDLC there wasn't any planning for software security. This issue can cause bugs, more development effort, and delayed deployments. That's why adapting security concepts to the SDLC is necessary. So, to avoid these kinds of issues inside the SDLC, the developers should follow security assurance activities like code reviews during build and review, penetration testing and architecture analysis during the design phase [32].

- Weaknesses of secure software

The developer must have a thorough awareness of the system's weaknesses when it comes to a software vulnerability. As for the following research, there are 10 common software vulnerabilities according to Stuart Foster's article [33]. They are Injection attacks, Broken Access Control, Insecure Design, Security Misconfiguration, Identification and Authentication Failures, Vulnerable and Outdated Components, Software and Data Integrity Failures, Cryptographic Failures, Security Logging and Monitoring Failures and Server-Side Request Forgery.

- Secure Coding Practices and problems of them

According to SEI CERT Coding Standards, there are top 10 ways of secure coding practices [34]. Every developer should follow these best practices.

1. Validate Inputs.

Validate all input that the system takes from unknown data sources. Proper input validations help to avoid most of the security vulnerabilities. Also, developers should be cautious about external data sources too for example user-controlled files, environment variables, network interfaces and command line arguments.

2. Heed compiler warnings

By using static or dynamic analysis tools like SonarQube developers can detect and modify the vulnerabilities and the using highest warning level compiler developers can find the flaws in code and fix them. This can save the developed system from most security vulnerabilities and make the developer a better developer.

3. Architect and design security policies

When initially designing the system designers should implement and enforce the security policies of the system. That will be helpful for managing security levels throughout the system.

4. Keep it basic

Ensuring that the system architecture is as straightforward and compact as feasible. If the design of the system is more complex it can contain more errors and when comes to fixing those bugs and ensuring the security of the system will take more effort will be used to resolve them.

5. Default denies

In the default system, all the security access is denied and only the responsible use of the system can allow access to the system.

6. Follow the principle of least privilege

The bare minimal privileges required to complete each procedure should be used. Only use any additional privileges for the time required to finish the complex task. This method lowers the possibility of an attacker executing arbitrary code with increased privileges.

7. Adopt a secure coding standard

Using the necessary secure coding standards for the developing system. These standards should be relevant to the development language of the system.

8. Sanitize data sent to other systems

Attackers can invoke attacks on the developed system and destroy the using injection attacks. To avoid these kinds of attacks, developers must sanitize all the data passing throughout the systems like command shells

9. Extensive defensive drills

Create the system with numerous defensive measures so that it will be protected in layers, and if the first layer fails, the second will continue to protect it and lessen the attack's harm.

10. Use effective quality assurance techniques.

Occasionally, vulnerabilities are found and fixed utilizing effective quality assurance processes. Fuzz testing, source code audits and penetration testing are all essential components of an effective quality assurance program. It's possible that systems that have passed independent security audits are more secure. For instance, when identifying and rephrasing inaccurate remarks, external reviewers offer a different perspective.

IV. Discussion & Analysis

The many SDL stages, which are simple for developers with varied IT backgrounds to follow and serve as a guide to help developers to maintain the security of the developing system as a top priority, are one of the important issues mentioned in the literature in this article. In order to give developers better-defined security and, ultimately, compliance with the project objectives, the first stage of the SDL focuses on the idea of the system, such as what it is used for and who its target users are. Concept and planning, the first stage of SDL, involves a number of advised methodologies. After the SDL discovery technique previously described the second stage in this phase is the definition of systems security and compliance objectives. Another recommendation made by this practice is to use an SDL methodology for the plans' layout, especially for those that involve SDL activities.

Before beginning any project, the SDL discovery process is crucial because it enables developers to identify security problems as they arise so that the development team can quickly deliver a fix. Another well-known initial phase of the SDL process is security requirements. Listing down prospective security needs, whether they be for front end-end or the back end, as the term suggests, helps developers find potential issues and fix them as soon as possible, particularly in non-compliant portions of the developing system [15]. Security awareness training is an essential component of this phase's final practice since technology is always changing and senior developers should be informed of any modifications to the SDL models that typically improve their security and project planning [15]. The discipline of software development can be decentralized or centralized. Component-based software development and distributed software engineering have become more popular in recent years with the growth of the internet and online communication technologies [16] have also grown in popularity, which has led to the emergence of new problems [17].

The following SDL phase is also referred to as design and architecture and includes the three practices of threat modelling, secure design, and third-party vulnerability tracking. This phase focuses on the creation of a product that conforms to the requirements using threat modelling, which assists in the detection of potential cyber-attack schemes. Developers may propose solutions for the application while it is still in the design process, which saves money and time while also resulting in the provision of an incident response strategy for prospective dangers. The next vital method of the second phase is secure design, which emphasizes documenting the design and any adjustments afterwards before they are put into practice to detect security issues and enable the adequate assessment and re-evaluation before the system is made accessible to the public [18].

Third-party software tracking, which is the final procedure in this stage, focuses on security vulnerabilities as well because they can jeopardize the system's integrity. Therefore, regular maintenance and cautious third-party access monitoring are required to strengthen and improve the system's security integrity. Using three methodologies—scanning tools, secure coding, and code review—the project is carried out and the system is put up during implementation, the third step of the SDL process. The third step of secure coding largely focuses on the system's or project's back end as a reminder to developers or debuggers to utilize phrases that are encrypted in the source code of the system to reduce vulnerabilities [19].

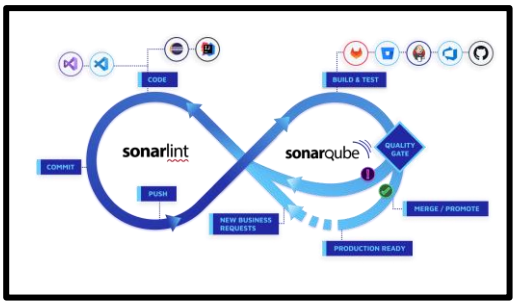

Figure 2: SonarQube Cycle

In order to completely identify the risk, it is suggested to scan every day from the beginning of the project's development until it is put into use. [20]. In order to find security problems that do not require the system to be running, scanning tools like Static Application Scanning Tools (SAST) automatically go through the source code. The last step in this process is the code review. Even while SASTs and other tools may expedite the process, it is still important

to frequently review the code by hand after each tool is used to conduct a double-check for security issues that tools cannot detect.

Three best practices are included in testing and issue remediation, the fourth stage of the SDL. These include penetration testing, dynamic scanning, and dynamic application security testing (DAST). The first step in this phase is dynamic scanning, which involves the project's implemented system while it is in use. Before the system's security is formally implemented, developers can have a better understanding of it by watching how it is been configured. An ongoing simulation of what cybercriminals would use is shown here [21]. Penetration testing, often known as Pentest, is essentially white-hat hacking done by other competent professionals to gauge the system's dependability. The fuzzing test is moreover one of the suggested procedures for the fourth step. By making the system resistant to attack types like SQL injection after processing it with a series of randomly generated inputs, this approach helps to increase system protection. [22].

A reaction plan and security checks are two essential phases in the deployment and maintenance phase of the SDL process, which is the fifth phase overall. Since it specifies activities to take in the event of a cyberattack and aids the monitoring team in resolving problems promptly, incident response planning is one of the subjects covered in practice at this stage. The second method is daily security tests that are continuous and help to maintain the security integrity of the system. This makes it possible to identify any weaknesses and gives the incident response team the ability to act fast by utilizing process strategies [23].

The implemented system no longer receives updates, and the data gathered throughout its lifespan is properly disposed of. This is when the implemented system enters the sixth phase of SDL, known as "end-of-life support." In the event that the system must be shut down altogether, there are two distinct processes that deal with the proper disposal of raw data. Data retention is the initial step in this last stage. This is the continued preservation of data despite not receiving updates, as required by firm retention policies for compliance and adherence to the legal process of user's data protection and to avoid pointless litigation from customers or users.[24]. The second procedure for this phase is data disposal, wherein a system is developed expressly for getting rid of undesirable data while paying meticulous attention to details, and archived data must be safeguarded with encryption to enable a secure manner of retrieval for further usage. [24].

V. LIMITATION AND RECOMMANDATION

However, the SDLC models may have drawbacks, just like any other software development methodology, which may have a significant impact on the client's decision. The software development lifecycle system's potential limitations include increased project development costs and duration, low flexibility, high amount of client participation, and testing may prove to be too difficult for certain development teams to handle [44]. These limitations may happen due to what type of SDLC model has been chosen for a software development project. Each SDLC model comes with its own advantages and disadvantages, even in terms of the security aspect. Since it takes longer and costs more money to detect and correct issues after a release, testing is typically put off until the last minute. The truth is that a secure software development lifecycle doesn't have to be cumbersome or expensive, and making your SDLC does not imply compromising security [45]. A major security concern is striking a balance between risk and resource constraints. The first challenge was characterizing the risk associated with the various products, which may be challenging.

The Agile SDLC model promotes continuous iterative development and testing throughout the development lifecycle. In the agile technique, tasks are divided into time boxes to provide specified features for a release. Having a team that is given the authority to include security into their procedures, processes, and pipelines is essential for security in Agile software development. Agile's key advantage is the ability to produce valuable, user-ready software in a limited number of iterations. Based on what you learnt from the previous iteration, you incorporate improvements with each new one [46]. When incorporating security into Agile Software Development, the Agile team must address several challenges: the stress of quick iterations, knowledge gap in information security, insufficient security awareness and the compatibility of agile approaches and security activities. Extreme Programming (XP), Scrum, and Dynamic Systems Development Method are some of the different ways to integrate security in Agile methodology.

The waterfall technique was widely used in the development of software up to the year 2000. Even after the Agile Manifesto was published in 2001, many firms continued to use the Waterfall technique for the previous 10 years. Because this methodology is not suitable for long and ongoing projects, and when the risk of requirements change is moderate to high. However, there are specific situations in which the waterfall concept is still advised. Imagine a system where human life is at risk and one or more deaths might result from a system failure. Alternatively, picture a system where the safety of people comes above worries about time and money [47]. Therefore, the Microsoft Security Development Lifecycle and the NIST framework for Security Considerations in the System Development Life Cycle are two examples of suggested and "best practice" frameworks that may be used to assure safe development with the Waterfall methodology [48]. Security training is the first stage of Microsoft SDL, and it highlights how crucial it is for programmers and other team members to have the right

security knowledge for the position. Every phase of the waterfall model in Microsoft SDL, including the planning phase, requirements phase, and design phase, includes security-related tasks. For instance, risk analyses are a part of the design process. The majority of security problems occur during coding. As a result, it is crucial that the programmer generally follow strict and secure programming techniques.

In [26], research participants tried a few different tactics, including automated tools and procedures to decrease their labour. Additional automation was frequently mentioned as a potential way to enhance application security. The fact that many of the current procedures and instruments do not lend themselves well to automation was also mentioned by attendees. Auditor awareness of the value of human judgment in security choices was another factor.

VI. Proposed Approach to Solve Security Issues

In order to increase security within a software development lifecycle, it is necessary to consider both social and technological factors. Culture has a significant impact in software development processes. For the organizational structure to create and maintain a positive and healthy security culture, security must be included as a natural part of developer routine and the final product. It seems reasonable that any adjustments to structures or procedures to account for cyber security would be influenced by developers' emotional states, which also have an impact on the effectiveness of new structures and procedures. For instance, the perception of an agile shift by software engineers would rely on difficulties with teamwork, communication, commitment to change or tolerance to it, trust, and the company's environment. Any security intervention must take into account how to ensure that software engineers have autonomy, are empowered in their job, and can still balance cooperation and coding.

While looking into the technical part to find a solution, all Agile Methodologies adhere to similar fundamentals, they nonetheless have unique security integration strategies. Three agile software development techniques—Scrum, Extreme Programming, and Dynamic Systems Development Method—are the topic of this section. Each technique has its own proposed solution to integrate security into the SDL.

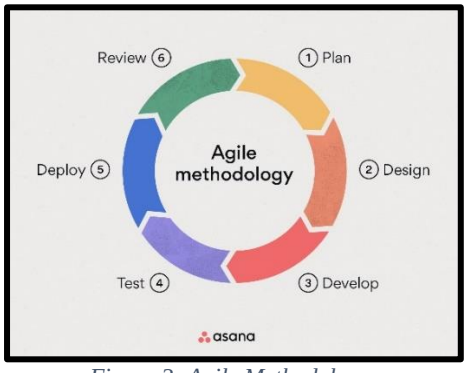

Figure 3: Agile Methodology

Veracode proposed the Security Sprint Technique and the Every-Sprint method as two ways to incorporate security into the Scrum framework. The every-sprint techniques involve the security user stories in each Sprint. The most expensive requirement for a security professional on the Scrum development team is the biggest problem with this strategy. The Security Sprint process involves evaluating and creating security user stories in a distinct Sprint. This approach could slow down the development process.

Secure Scrum is the suggested safe technique for using Scrum. The development of safe software across the whole software development process is the main emphasis of this. The four components of Secure Scrum are Identification, Implementation, Verification, and Definition of Done. These four elements will be combined with six Scrum pieces to increase software development security. Secure Scrum is built on the use of S-Tag and S-Mark. The identification component uses the user stories of the product owner and stakeholders to pinpoint security issues. To ensure that the development team is aware of security risks, employ the implementation component. The verification part ensures that the team member looked into the security problems with the S-Mark tasks. The verification element is part of the definition of done. The verification component is managed at the daily scrum meeting. The Definition of Done component makes sure that either internal or external resources must be used to carry out the security verification. During the development process, team members' understanding of security is being raised and providing the resources for external security as needed are some of the advantages of Secure Scrum.

Inbuilt Security and Role-based Extreme Programming are the suggested secure methods for implementing Rewards. Adding security components to the Extreme Programming framework is useless without a security expert. Extreme Programming now has a new post called "Security Master." A Security Master is in charge of the security of the created program. A programmer who focuses on security issues is known as a security master. Inbuilt security is another method for incorporating security into extreme programming. The goal of the Inbuilt Security approach is to integrate security-related tasks into extreme programming procedures. This strategy omits to mention the new team member. The intense programming team might not be able to finish security-related tasks without a security expert. Additionally, some extreme programming techniques are incompatible with the needs of security-related operations. The new position of Security Master in Role-based Extreme Programming is highlighted by this discipline. The security master's abilities have a significant impact on the product's security. A hole in the product's security is inevitable if the security master is untrained.

To solve the security issue in the waterfall methodology, using STRIDE and DREAD methodology is the recommended way. The waterfall model is divided into Requirements, Design, Implementation, Testing, and Release/Maintenance. The requirements stage focuses on defining a description of the software to be produced and its

necessary features. It is meant to be a point of agreement between the clients and the software developers over the precise tasks that must be completed. Making security an explicit requirement is the best method to ensure that it is given top attention in any software development project. STRIDE is a threat modelling approach that trains programmers to think like attackers in order to spot possible misuse scenarios for their applications. They look for potential attack vectors that fall under the following categories: mimicking, manipulation, repudiation, data leak, denial of service, and privilege elevation. The purpose of the exercise's threat modelling (STRIDE) component is to go through these categories and find any potential dangers that the application may be exposed to in each one.

Following the identification of all potential risks, the risk analysis part (DREAD) may be carried out to determine which dangers are most important, allowing for the inclusion of controls against these threats in the security requirements. Based on damage capability, reproducibility, vulnerability, affected victims, and visibility, DREAD rates hazards. Then, on a numerical scale of 1 to 3, with 1 denoting a danger with little impact and 3 denoting a major threat, each threat is ranked in each category. The total risk score for that specific danger is then calculated by averaging the results for each category. The design phase, which will lay out the application architecture and provide a foundation for the software's implementation, should be started as soon as the requirements are finished. What security measures should be established and how these controls are to be applied should be explicitly stated in the design phase's final output. The implementation phase might start once the design phase is finished. The actual writing of the code is the focus of this stage. Secure coding standards should be established inside the business before any code is ever created, and developers should be made aware of the significance of these standards and the necessity of adhering to them.

Fuzzing is a technique used to test the security of applications. Fuzzing involves feeding a large number of randomly generated inputs into an application to try to trigger all potential code paths and see if any problems arise as a result, such as application crashes, bypassing escaping, and so forth. This is done because inputs that cannot be handled correctly have the potential to be used as attack vectors. It's crucial to remember that an application will still have flaws no matter how meticulously safe SDLC processes were followed or how completely it was tested. Prior to the launch of the software, we should have procedures in place for identifying and verifying security flaws and other types of software problems, as well as for fixing such flaws.

VII. Conclusion

Unquestionably, a vast number of software programs are created every day, and everyone from small-business owners to human users at home depends heavily on them to carry out their everyday responsibilities. In this scenario, every software development body and software product evaluation commission must prioritize the secrecy, safety, and protection of every software program. In reality, software developers did not concentrate their efforts to protecting software programs in the past, or even now; rather, they provided only the minimal amount of security required to execute the software.

As we are aware that prevention is preferable to cure, the cost of building security measures is recognized to be much cheaper than the cost utilized to combat an assault. In this regard, existing research articles have been evaluated. To inform the necessary stakeholders of the security implications of a software development process, a full discussion of each stage of the SDLC and SSDL is conducted. After performing the inspection and analysis described in this article, we came to the conclusion that security measures may be deployed concurrently in each phase of the SDLC.

Risk analysis, creating threat models, code review using bug scanning tools, and system testing are a few of them. Each security measure has a purpose in identifying the threat and vulnerabilities a system may have. The worst-case situations that might occur in the future and some mitigation strategies have also been discussed. Despite having every security mechanism, it could not ensure zero vulnerability due to the rapidly evolving technical environment. The suggested method includes routinely updating the installed software system, using security scanning tools, and regularly training stakeholders.

VIII. References

[1] C. Freeman, "Secure SDLC 101," Synopsys, 08 August 2022. [Online]. Available: https://www.synopsys.com/blogs/software-security/secure-sdlc/.

[2] H. S. T. R. B. K. Edyta Karolina Szczepaniuk, "Information security assessment in public administration," *Computers & Security,* vol. 90, p. 101709, 2020.

[3] H. Assal and S. Chiasson, "Security in the Software Development Lifecycle," *Fourteenth Symposium on Usable Privacy and Security,* pp. 281-296, 2018.

[4] M. Saito, A. Hazeyama and N. Yoshioka,, "A Case-based Management System for Secure Software Development Using Software Security Knowledge," *Procedia Computer Science,* vol. 60, p. 24, 2015.

[5] M. H. J.-R. and M. J. , "The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies," *Electronics,* vol. 1218, p. 8, 2019.

[6] K. Rindell, J. Ruohonen and J. Holvitie, "Security in agile software development: A practitioner survey," *Information*

and Software Technology, vol. 131, no. 106488, p. 13, 2021.

[7] C. Weir, I. Becker and L. Blair, "A Passion for Security: Intervening to Help Software Developers," p. 10, 2021.

[8] D. Ashenden and G. Ollis, "Putting the Sec in DevSecOps: Using Social Practice Theory to Improve Secure Software Development," *NSPW '20: New Security Paradigms Workshop 2020*, no. https://doi.org/10.1145/3442167.3442178, pp. 34-44, 2020.

[9] A. V. D. Heijden, C. Broasca and A. Serebrenik, "An empirical perspective on security challenges in large-scale agile software development," *ESEM '18: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, no. https://doi.org/10.1145/3239235.3267426, p. 1–4, 2018 .

[10] T. W. Thomas, M. Tabassum, B. T. Chu and H. R. Lipford, "Security During Application Development: an Application Security Expert Perspective," *CHI '18: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, no. https://doi.org/10.1145/3173574.3173836, p. 1–12, 2018.

[11] J. C. S. Núñez, A. Caro and P. G. Rodriguez, "A Preventive Secure Software Development Model for a Software Factory: A Case Study," pp. 1-1, 2020.

[12] R. A. Khan, S. U. Khan and M. Ilyas, "Exploring Security Procedures in Secure Software Engineering: A Systematic Mapping Study," *EASE '22: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, p. 433–439, 2022.

[13] H. AL-MATOUQ, S. MAHMOOD, M. ALSHAYEB and . M. NIAZI, "A Maturity Model for Secure Software Design: A Multivocal Study," vol. 8, p. 19, 2020.

[14] R. A. Khan, M. Y. Idris, S. U. Khan, M. Ilyas, S. Ali, A. U. Din, G. Murtaza, A. W. Khan and S. U. Jan, "An Evaluation Framework for Communication and Coordination Processes in Offshore Software Development Outsourcing Relationship: Using Fuzzy Methods," *IEEE Access*, vol. 7, pp. 112879-112906, 2019.

[15] M. D. Roshaidie, W. P. Han Liang, C. G. Kai Jun, K. H. Yew and . F.-t.-Z. , "Importance of Secure Software Development Importance of Secure Software Development," p. 13, 2020.

[16] C. Diwaker, T. P. S. A. N. A. Z. N. A. A. and S. M. , "A New Model for Predicting Component-Based Software Reliability Using Soft Computing," vol. 147191, p. 13, 2019.

[17] W. S. H. B. Z. N. H. M. and A. N. , "Improving Knowledge Sharing in Distributed Software Development," *International Journal of Advanced Computer Science and Applications*, vol. 10, p. 10, 2019.

[18] I. C. "Plan, Excercise, Train.," 2022.

[19] "SonarQube Documentation," sonarQube, 2008. [Online]. Available: https://docs.sonarqube.org/latest/.

[20] s. "CS302: Software Engineering," saylor.org, 2010. [Online]. Available: https://learn.saylor.org/.

[21] H. Goslin, "Why is Dynamic Analysis an Important Part of Your AppSec Mix?," veracode, 31 july 2020. [Online]. Available: https://www.veracode.com/blog/intro-appsec/why-dynamic-analysis-important-part-your-appsec-mix.

[22] s. "Continuous Penetration Testing can reduce exposure time and prevent breaches.," sprocketsecurity, 2022. [Online]. Available: https://www.sprocketsecurity.com/.

[23] T. "Test Environment Management 10 Essential Practices," Tem.com, 12 November 2019. [Online]. Available: https://www.testenvironmentmanagement.com/test-environment-management-10-essential-practices/.

[24] B. PoseyPaul and C. A. , "data retention policy," .techtarget, 2018. [Online]. Available: https://www.techtarget.com/searchdatabackup/definition/data-retention-policy.

[25] "Secure Software Development: Best Practices, Frameworks, and Resources," hyperproof, 28 April 2022. [Online]. Available: https://hyperproof.io/.

[26] M. Zhang, X. de Carné de Carnavalet and L. Wang and A, "Large-Scale Empirical Study of Important Features Indicative of Discovered Vulnerabilities to Assess Application Security," *Transactions on Information Forensics and Security*, vol. 14, p. 2330, 19.

[27] J. C. S. Núñez, A. C. Lindo and P. G. Rodríguez, "A Preventive Secure Software Development Model for a Software Factory: A Case Study," vol. 8, pp. 77653-77665.

[28] J. Li, Y. Zhang, , X. Chen and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," vol. 72, pp. 1-12, 2018.

[29] A. Misra, "Aspects of Enhancing Security in Software Development Life Cycle. International Journal for Computational Methods in Engineering Science and Mechanics," vol. 1, pp. 203-210, 2017.

[30] S. S. M. A. H.-L. C. T. Wael Khreich, "Combining heterogeneous anomaly detectors for improved software security," *Journal of Systems and Software*, vol. 137, pp. 415-429, 2018.

[31] P. R. Khan, "Secure software development: A prescriptive framework. Computer Fraud & Security," pp. 12-20, 2011.

[32] S. K. Amit Kumar Srivastava, "An effective computational technique for taxonomic position of security vulnerability in software development," *Journal of Computational Science*, vol. 25, pp. 388-396, 2018.

[33] Y. L. a. G. Lee, "HW-CDI: Hard-Wired Control Data Integrity," vol. 7, pp. 10811-10822, 2019.

[34] M. Siavvas, E. Gelenbe, and D. Kehagia, "Static Analysis-Based Approaches for Secure Software Development," *Communications in Computer and Information Science book series*, vol. 821, 2018.

[35] M. N. M. J. N. e. a. Humayun, "Cyber Security Threats and Vulnerabilities: A Systematic Mapping Study," *A Systematic Mapping Study. Arab J Sci Eng*, vol. 45, p. 3171–3189, 2020.

[36] S. J. H. N. J. a. M. H. K. Hussain, "SYN Flood Attack Detection based on Bayes Estimator (SFADBE) For MANET," *International Conference on Computer and Information Sciences (ICCIS)*, pp. 1-4, 2019.

[37] L. Elliott and R. Mojdehbakhsh, "A Process for Developing Safe Software," p. 6, 1994.

[38] S. Al-Amin, N. Ajmeri, H. Du, E. Z. Berglund and M. P. Singh, "Toward Effective Adoption of Secure Software," p. 32, 2018.

[39] Z. Shen and S. Chen, "A Survey of Automatic Software Vulnerability Detection, Program," *Hindawi Security and Communication Networks*, vol. 2020, p. 16, 2020.

[40] H. Assal, S. Chiasson and C. U. , "Security in the software development lifecycle.," *Fourteenth symposium on usable privacy and security*, pp. 281-296, 2018.

[41] Microsoft, "What are the Microsoft SDL practices?," Microsoft, [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/practices. [Accessed 2022].

[42] C. Freeman, "Secure SDLC 101," Synopsys, 2022. [Online]. Available: https://www.synopsys.com/blogs/software-security/secure-sdlc/. [Accessed 2022].

[43] S. Foster, "Vulnerabilities Definition: Top 10 Software Vulnerabilities," Preforce, 7 July 2020. [Online]. Available: https://www.perforce.com/blog/kw/common-software-vulnerabilities. [Accessed 2022].

[44] R. Seacord, "Top 10 Secure Coding Practices," Carnegie Mellon University, 02 May 2018. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices. [Accessed 2022].

[45] Intellectsoft, "System Development Life Cycle: Basics to Know," Intellect soft, [Online]. Available: https://www.intellectsoft.net/blog/what-is-system-development-life-cycle/. [Accessed 2022].

[46] Veracode, "SDLC Agility & Security," Veracode, 2022. [Online]. Available: https://www.veracode.com/security/sdlc-agile. [Accessed 2022].

[47] N. Butler, "Security in Agile software development: a simple guide," Boost, 10 February 2022 . [Online]. Available: https://www.boost.co.nz/blog/2022/02/security-in-agile-software-development. [Accessed 2022].

[48] Try Qa, "What is Waterfall model- Examples, advantages, disadvantages & when to use it?," Try Qa, 2022. [Online]. Available: http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/#Testing. [Accessed 2022].

[49] A. Ø. M. O. E. Jøsang, "Cybersecurity Through Secure Software Development.," *WISE 2015. IFIP Advances in Information and Communication Technology*, vol. 453, p. 53–63, 2015.