ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

AN INSIGHT INTO HYBRID AGILE SOFTWARE DEVELOPMENT APPROACH AND SOFTWARE SECURITY AMONG SOFTWARE ENGINEERS: A CRITICAL EVALUATION

SITI SARAH MAIDIN[1] , NORZARIYAH YAHYA[2]

[1]Senior Lecturer, [1]Faculty of Data Science and Information Technology, INTI International University, Nilai, Malaysia

[2]Senior Lecturer. Kulliyah of Information and Communication Technology, IIUM, Gombak, Malaysia

E-mail: [1]sitisarah.maidin@newinti.edu.my, [2]norzariyah@iium.edu.my

ABSTRACT

The aim of this paper is to provide an overview of the hybrid agile development approach specifically in software development. Different platforms were used to investigate the factors that influence developers in choosing the preferred model for software development. In addition, this paper also examines the security elements in a software development project. It identified 3 factors that motivate software developers to focus on software security in a project. These are the company's policies and culture, i.e., the overall company culture regarding security, the application domain, i.e., the developers' perception of the benefits of security for their applications, and finally, the use and complexity of security tools. This article is organized as follows. The first section is the introduction, which explains the software development methodology. The subsequent Materials and Methods section provides an overview of searching journals in various databases, including Google Scholar, IEEE, ACM, and Science Direct. This paper concludes with recommendations on other areas that can be explored in the area of software security in the context of the hybrid agile approach.

Keywords: *Agile, Hybrid Agile, Software Development, Software Security, Model*

1. INTRODUCTION

Software is extremely important, particularly in industry. As a result, software security has become an essential component and requirement in SDLC [1]. Traditionally, software security is added after the software development process is complete. Security is regarded as an optional feature in SDLC [2]. Secure software development methodologies are the procedures for achieving security goals through the design, construction, and testing of software. Frameworks for secure software development prioritize in integrating security in all phases of the software development life cycle (SDLC). Some well-known secure software development approaches are as follows:

- Secure Agile is an Agile software development extension that incorporates security practises into the Agile process.
- Secure DevOps: Continual security testing and delivery are prioritized in the Secure DevOps methodology, which combines the DevOps and security principles.
- Microsoft's Security Development Lifecycle (SDL): a thorough approach to secure software development that includes security testing, threat modelling, and training throughout the SDLC.
- The Open Web Application Security Project (OWASP): compiled a list of the top ten web application security risks, along with advice on how to reduce those risks.
- ISO/IEC 27034: a guideline for integrating security into the SDLC, including requirements, design,

implementation, testing, and deployment, for secure software development.

A secure software development methodology should be chosen based on the project's size and complexity, the type of software being developed, and the security resources available. It's crucial to pick a methodology that meets the requirements of the development team and the organisation, and to review and update it as necessary on a regular basis.

A software development methodology is a framework for defining how software is created, managed, and presented. Among the most popular software development methodologies are Scrum, Kanban, DevOps, Lean Software Development, Test-Driven Development (TDD), Feature-Driven Development (FDD), and the Spiral methodology. Agile methodologies have grown in popularity since a group of software practitioners published the philosophy for agile software development in 2001 [3].

Agile software development methodologies have provided significant benefits to software development organizations over traditional waterfall software development methodologies. Each methodology has distinct software development methodologies, strengths, and shortcomings. The selection of a methodology is influenced by a number of variables, including the scope of the project, the nature of the software being built, and the preferences of the development team. Although each agile methodology has numerous processes, theoretically, certain aspects of its core principles were sometimes overlooked during implementation [4]. The likelihood of resource exhaustion increases as the speed of agile development increases. SDLC's primary concern became software security. As a result, some organizations transition from the agile approach to the hybrid agile development approach, which combines the waterfall and agile models.

The development process includes new coding as well as the use of third-party components and libraries. The purpose of incorporating software security into the software development process is to detect security flaws and prevent defects. According to a study conducted by [6], the main reason organisations do not incorporate software security into their development process is a lack of security plan, knowledge, procedures, and resources, as well as dealing with competing priorities. This study investigates the organisational and human issues that contribute to insecure practises in software development methodology.

2. MATERIALS AND METHODS

In this study, a uniform search of keywords or key terms was defined. The keywords include "Hybrid Software," "Hybrid Development," and "Hybrid Agile," which were the main words of study for software security in hybrid agile software development. The most relevant papers for the study from different research databases for the previous twelve years (12) are listed in Table 1 and the outcome is the assessment from the combination of the research paper title, abstract, and keywords. The database used is Google Scholar, IEEE Explore, ACM, and also Science Direct. In total, 2044 numbers of articles were returned from the search. After the filter, only 196 articles are accepted for review in this paper. Table 1 indicates the search technique used to retrieve the relevant paper. There were 4 databases included in the search which are Google Scholar, IEEE Explore, ACM, and Science Direct. A total of 2024 search results were obtained for the search query:

"Hybrid Software" OR "Hybrid Development" OR "Hybrid Agile".

After the filter by Software Security or Secure Software, only 196 papers are included in the study. Duplicate papers found in the data collection from the search were removed. The article must focus on the study of secure software development methodology as the definition stated in the introduction and directly related to security practices. Table 1 shows the distribution of the articles.

Table 1: Relevant Papers included in the study (Jan 2010-Dec 2022)

	Search Query	**Filter**
Database	Hybrid Software OR Hybrid Development OR Hybrid Agile	Software Security OR Secure Software
Google Scholar	1550	149
IEEE Xplore	96	1
ACM	152	44
Science Direct	246	2
Total	2044	196

3. RESULTS AND DISCUSSION

Many SDLC processes or models have been discussed and presented by researchers such as waterfall, V-Model, and agile through comparison or systematic review. Many works of literature emphasize a secure development lifecycle. Some study highlights the current security initiative. Only a few of the literature focus on human factors, secure development practices, and factors influencing the adoption.

3.1 Traditional Model

The waterfall software development approach was first introduced by Bennington in 1956 and later improved by Royce in 1970 [7]. It created a foundation for software development where the requirement needs to be well-defined and analyzed before any design or development. Bennington's model included operational elements such as operational plan and operational specification as part of the model.

Royce improved the model by adding a feedback loop to revisit each stage of his model to handle unforeseen design difficulties. It is to ensure any redefinition of requirements is taken care of by design and validation (testing) stages [8]. Another aspect of the waterfall approach is that it required extensive documentation. However, the waterfall model is suitable for a small-scale project whereby all the requirements are well-defined. It can also cut down the operational cost whereby there is minimal possibility of revisiting the previous phase.

3.2 Agile Model

Agile can be defined as a development approach that is both flexible and innovative. It's a strategy for thriving during the chaos and unpredictability that often characterizes software development projects. The Agile Manifesto advocates for iterative and evolutionary approaches to software development, which shorten the development cycle and increase the quality of the result. As a lightweight process paradigm, agile places an emphasis on communication and collaboration between all stakeholders. Agile's primary intent is to shorten the duration of development cycles, increase software quality, and decrease overall costs. Agile encourages iterative and evolutionary approaches to development, which shorten the time required to both create and deliver a product. The flexibility to adapt to new circumstances and alter strategies is another perk.

Traditional software development approaches face numerous challenges due to rapid market changes. Agile methods were proposed to overcome the challenges. These methods allow the software engineering team to revolve around development rather than its design and documentation [9]. Agile principles are getting customer involvement throughout the development process, having incremental delivery, concentrating on people rather than procedure, embracing change, and maintaining simplicity [3]. Different agile approaches represent these principles in different ways. Agile is derived from the “agility” concept whereby it promotes flexibility, especially for integration purposes.

3.2.1 Extreme Programming (XP)

XP evolved due to the lengthy development cycles in traditional development methods. In 1996, Kent Beck created the XP model through his payroll project - Chrysler Comprehensive Compensation System (C3). The model was released in the Extreme Programming Methodology book in 1999. The approach was developed to push recognized good practices in development to the 'extreme' levels [9]. It promotes frequent releases in short development cycles. It focuses on lightweight processes. The main phase involved in the cycle is Planning, Design, Coding, and Testing. As an iterative model, the overall project is divided into small functions. The cycle of development for one feature is started from the design to the testing phase. After executing and debugging for one feature is done correctly, the programmer will move to the next feature. XP model required rapid release cycles in which continuous communication between stakeholders and developers [10].

3.2.2 Scrum

Scrum is a well-known Agile methodology for managing and finishing projects in the software development industry. It is founded on incremental and iterative development and places a strong emphasis on teamwork and adaptability. To provide the customer with incremental value, a cross-functional team works on sprints in Scrum. The group regularly reviews project goals and priorities in light of feedback from stakeholders and holds daily stand-up meetings to discuss progress and pinpoint any obstacles. Delivering high-quality software that satisfies the needs of the client is the objective of the Scrum methodology.

Scrum is more suitable for small software projects as it not as much emphasis on the process, which is happening in a large project (central command structure). In Scrum, development happens in a series of short sprints [8]. In the iterations (sprints), the task is decomposed and group in backlogs. Scrum ensures the processes are simple and effectively deliver small working software packages [11]. Scrum starts with gathering user requirements (user stories) in a product backlog. A sprint backlog is created based on this product backlog. Each sprint will have its development process. Scrum is having a daily scrum meeting to discuss the problems that arise and evaluate the progress in that sprint. After each sprint concluded, there will be a deliverable product to the customer [11]. Scrum is suitable when the scope of the project is not well-defined, and it also involved a large-scale of project. The uniqueness of Scrum is it allows the "back and forth" of the phases whereby there is a room for the improvement for the software developer to enhance back the outcome in each phase.

3.2.3 Iterative/Incremental Model

The iterative/incremental model is a software development strategy that includes numerous cycles of creation and enhancement. Every cycle, or iteration, entails finishing a small section of the project, assessing and testing the outcomes, and using the feedback to guide the following iteration. This method enables teams to adapt to altering requirements and make improvements gradually, resulting in the delivery of a usable product more quickly and with less risk. The process begins with a high-level comprehension of the issue at hand, and iterative cycles of planning, development, testing, and feedback are used to iteratively improve the solution. Teams can respond to changing requirements using this method and make necessary adjustments, resulting in a final product that better satisfies the needs of the client.

This model addressed the problem of time to deliver software products. Consecutively smaller releases are implemented instead of delivering an extensive system. There are many advantages as compared to the traditional waterfall approach [18]; for instance, requirements can be prioritized, and customers receive part of the system early. This model is a blend of both iterative design and incremental build model for development. During software development, there might be multiple iterations of the development cycle progress runs concurrently. The process is defined as an incremental build method. In the incremental process, the entire requirement is separated into many builds. In each build, the requirement, design, testing, and implementation phases will be gone through. Each release is adding functionality to the former release. The process remains until the whole system is completed as per the requirement.

3.3 Hybrid Agile Model

A hybrid approach can help organisations benefit from Agile and traditional project management methods while meeting their projects' and teams' needs. There are several reasons why the developers prefers hybrid. This includes (i) The company has various teams: Some teams prefer Agile, while others prefer traditional. The needs of different teams can be met with a hybrid approach. (ii) The organisation has a variety of project types: Some projects in an organisation may necessitate a more structured and predictive approach, whereas others may necessitate greater flexibility and agility. A hybrid approach can be used in such cases to tailor the project management process to the specific needs of each project. (iii) Some teams in an organisation may be more experienced and comfortable with a traditional approach, whereas others may prefer an Agile approach. A hybrid approach can be used to meet the needs of various teams and allow them to work in the most comfortable and effective way for them.

When developing software, a hybrid Agile model combines components from various Agile methodologies to produce a customised strategy that is tailored to the needs of a project or organisation. This model enables teams to take advantage of the advantages of various Agile approaches while making modifications to suit their particular circumstances. For instance, a hybrid Agile model might combine Lean for streamlining delivery processes, Scrum for project management, and Kanban for visualising workflows. The secret is to apply the Agile principles to the context of each project and to continuously assess and improve the methodology to get the best outcomes.

Hybrid software development is an approach that consists of any combination of traditional (plan-driven) and agile methods that an organization customizes or adopts to fulfill its needs such as culture, organizational structure, and application domain. Therefore, it is also called Hybrid Agile software development approaches. These approaches are the outcomes of a natural growth-driven process by learning, experience, and practicality [12]. The approaches are not dependent on the organization's size and external triggers [13]. According to the study done by [14], 76,8% of organizations implemented hybrid methods. Organizations utilize multiple frameworks, practices, and methods to provide efficient and effective development processes. It is for continuous produce software products that meet stakeholders' requests and market needs [14].

There was a project called HELENA (Hybrid and Development Approach in software systems development), which investigates the reality that organizations are using Hybrid methods. It is an international exploratory research project with multiple stages. According to [15], who are doing part of the HELENA project, most of the organizations use hybrid approaches goal is enhancing the frequency of delivery to clients, the flexibility and adaptability of the process to respond to change. HELENA project aim is to maximize productivity. In another part of HELENA's project, [16] found that different methods, practices, and frameworks have been used in combination as hybrid methods. It is happening to the organization of all industry sectors and sizes. They also discovered that the practices have restricted dependencies to the methods in hybrid. Hybrid inherited the concepts of combination. For instance, hybrid agile is an inherited based on “agile+non-agile” technique and method. The pro and cons of each technique and method are used to overcome the strength and weakness of each other. This where the unique of hybrid-agile is highlighted.

3.3.1 V-Model and Scrum

The hybrid software development approach was already in the market a long time ago. The V-model was first presented in 1991 by NASA. The model is different from the waterfall model, where its V shape folded in half at the lowest level of decomposition. The left leg of the V signifies the progress of user requirement into smaller details via the decomposition and definition process whereby the right leg signifies the verification and integration of the system components into the subsequence level of implementation [8]. The "development process" is located on the left side of the V, and the "verification process" is located on the right side.

The activities of requirements gathering, design, implementation, and testing are all a part of the development process. The activities of system testing, integration testing, and acceptance testing are all a part of the verification process. The point of the V, where the finished product is delivered to the customer, is where the two processes combine. The V Model places a strong emphasis on the value of exhaustive testing and verification at every stage of the software development process. Early defect detection and correction are desired to prevent more costly corrections. This can speed up development and cut costs while also enhancing the quality of the finished product.

Later in 2011, [17] presented a hybrid approach that combined Scrum into the V-model with high-level waterfall and low level agile. They proposed this hybrid V-model with three aspects of benefit. First, the requirements can be stated by the customer and project’s steam in the "waterfall-up-front" technique, which can decrease the risk of confusion in objectives and during project deliverables. Second, use Agile approaches in design, testing, and implementation that fast-track the process by reducing the risk of rescheduling, postponement, and the likelihood of rework. Then, the "waterfall-at-end" technique can be used, complete the entire acceptance process which is getting the project’s team and customer to perform high-level testing.

4. SOFTWARE SECURITY

Secure software lifecycle processes are a collection of practises and guidelines that organisations use to guarantee the security of their software systems at every stage of development, from design to maintenance. The intention is to reduce the likelihood of security flaws and safeguard the software against attacks. It is a lifecycle processes are “proactive methods that build security into a product by treating the poorly designed software at the source rather than applying patches as a reactive solution”. These processes put security into the full product development process, and include technology and people to avoid software security problems [5].

Security requirements gathering and analysis, Threat modelling and risk assessment, Secure Design and Architecture, Secure Coding Practices and Code Review, Penetration Testing and Security Testing, Security Monitoring, and Incident Response are a few of the key activities in secure software lifecycle processes. Many organizations have proposed integrated security in the SDLC. Microsoft introduced one of the first initiatives in 2004.

4.1 Security Development Lifecycle (SDL)

A thorough software development process that incorporates security into each stage of the software development life cycle is called the Security Development Lifecycle (SDL). Through the use of a structured and methodical approach to software development, the SDL is intended to assist organisations in producing secure software. Threat modelling and risk assessment, Security design, Secure coding and testing, Release process, and Post-release process are the typical phases of the SDL. Specific security tasks and products are included in each SDL phase, including security requirements, threat models, code reviews, penetration testing, and incident response planning. In order to lower the risk of security flaws and boost the overall security of the software, it is important to make sure that security is taken into account and integrated into every stage of software development. Organizations that create sensitive or important software systems, such as financial, medical services, and state institutions, frequently use the SDL to guarantee the security and integrity of their software and the data it handles.

Microsoft SDL released and became a mandatory policy at Microsoft in 2004. It was released to the public in 2006 with a 13-stage security development lifecycle process [19]. Microsoft's SDL currently has 12 security practices. The practices have been evolved according to the rising of cloud computing, the Internet of Things, mobile, and artificial intelligence. The practices cover a wide range of subjects, from threat modeling to security testing and managing the security risk of using third party components [20]. The first practice concentrates on providing security training to parties such as developers, service engineers, and production managers in the development process. Whereas the final practice is to establish standard procedures in responding to security incidents. The remaining practices from second to eleventh practice cover the main processes of the development lifecycle [20].

4.2 Open Web Application Security Project (OWASP)

OWASP, a nonprofit organisation, offers resources and tools to assist businesses in creating secure web applications. By increasing public awareness of typical security risks and offering advice on how to avoid them, OWASP aims to increase the security of software. Web application security resources from OWASP include OWASP Cheat Sheets, OWASP Tools and OWASP Chapters. Web app security practitioners, developers, and organisations all over the world use the resources provided by OWASP, which is a recognised and well-known source in the field.

OWASP is an international not-for-profit organization concentrated on refining the security of software. This organization aims to make software security observable for individuals and organizations to make a knowledgeable decision [21]. OWASP has many projects that demonstrated strategic value in terms of application security [22], [23]. For example, the OWASP SAMM project is providing an open framework that assists an organization in formulating and implementing a strategy for software security that is specific to an organization [22]. Whereas the OWASP Security Knowledge model is a tool given to use as a guideline for verifying and building secure software as well as providing training to developers about application security [23].

4.3 Building Security in Maturity Model (BSIMM)

An organization's software security practises can be evaluated and improved using the Building Security in Maturity Model (BSIMM), which is a data-driven framework. BSIMM was developed by a group of security professionals and is based on observations of actual software security initiatives at top companies. BSIMM is a collection of procedures and techniques that are frequently used in effective software security projects. Higher levels of maturity indicate a more sophisticated and all-encompassing approach to software security. These activities are arranged into a maturity model. In order to develop and enhance their software security programmes, organisations can use the BSIMM as a roadmap. It is a data-driven framework. BSIMM's software security framework comprises of 4 domains – Governance, Intelligence, SSDL Touchpoints, and Deployment that hold 12 practices that having three practices in each domain [24]. Each practice has its related activities. There are 119 activities in BSIMM entirely.

5. HUMAN FACTORS IN THE SECURITY OF THE SOFTWARE ENGINEERING PROJECT DEVELOPMENT

The security of software engineering project development is significantly influenced by human factors. Some of the factors are: Knowledge wherby the developers need to be knowledgeable about the value of security and the potential repercussions of security flaws and developers must receive instruction in secure coding techniques and in identifying and mitigating security risks, processes which means that there is a need to set up secure development processes and regularly evaluate them, culture that values security at every stage of development and is security-conscious. Collaborative effort can aid in identifying and addressing security risks throughout the development process and the development and security teams should work together to this end.

In the software development lifecycle, multiple stages require different cognitive processes. Developers frequently switch from task to task. Therefore the tools and methodologies should reduce unexpected interruptions and provide support to developers in focusing on their on-hand tasks. According to [25], many factors affect developers in practices and strategies they employ throughout their working day. Individual characteristics such as goals, working style, attitudes, and skills could influence the developer's strategy for implementing some functionality. Other factors that influence developers' approach such as type of development process, the developers' knowledge of the application, the context of use for the application that they are implementing and skills in using the particular programming language.

Software developers rely on different resources to gain the knowledge that they needed to perform their tasks. They usually rely on training, online information, or their experienced colleagues to specify questions and solutions. The developer spends a quarter of their working day in collaborating activities such as answering emails, attending the planned meeting, and informal meetings with colleagues. The other quarter of a working day is on coding related activities such as browsing the code related information [26].

5.1 Abilities and Expertise of Developers

A program's security relies heavily on the developer's skills and knowledge, making them essential to the process. In order to create trustworthy programmes, programmers need to possess certain skills and knowledge which includes:

- Secure Coding Knowledge: Understanding of secure coding practises and familiarity with common security vulnerabilities and how to circumvent them are two facets of this knowledge.
- Threat modelling: Potential security risks can be identified and countered with the help of a model of the threat landscape.
- Security Testing: Ability to integrate security testing into the development process, as well as familiarity with security testing methodologies and tools.
- Cryptography: Knowledge of cryptography and encryption algorithms, as well as the ability to employ these effectively to safeguard data.
- Incident response: The desire to learn new things and constantly better one's security knowledge and abilities.
- Continuous learning: Developers who possess these skills and knowledge are better able to produce secure software, to spot security flaws early in the development process, and to ensure that their products adhere to all applicable security standards.

There was an assumption that if software developers learned and cared more about security, they would able to prevent vulnerabilities. However, some studies argued that the increase of vulnerabilities could be due to the nonexistence of security guidelines or not mandated by the organizations [6], [27]. Other findings are to do with the lack of ability or expertise of developers to identify the vulnerabilities despite having general security knowledge [28], [29]. According to [29], two main influential factors impact the correctness of developers classifying the vulnerabilities positively. The factors are developers' prior security knowledge and experience with code analysis tools. They also found that the developers' overall experience did not have the anticipated positive impact on the correctness of vulnerabilities classification.

[28] argued that security vulnerabilities are the "blind spots" in developers' decision making processes. Their study showed that security is not commonly counted in as part of developers' programming tasks. However, when they are clearly warned about the potential security issue, the developers were more conscious of security in their tasks. Both [29] and [28] argued for a more personalized security experience for developers. [28] suggested for in-context security education rather than teaching developers about security vulnerabilities in general.

6. SECURITY TOOL ADOPTION

The process of incorporating security tools and practises into the coding process of software development in order to reduce the risk of security vulnerabilities and improve the overall security of the code is referred to as security tool adoption. Using secure coding guidelines, incorporating security testing into the development process, and using code analysis tools to identify potential vulnerabilities are all examples of this. Developers can write more secure code, detect and address potential security issues early in the development process, and help ensure that the software they create meets security requirements by using security tools and practises.

While investigating the integration of security tools in organization processes, researchers found that developers, in general, shown the attitude of 'security is not my responsibility' [6], [27]. According to [27] study, only a small number of large organizations were using security tools to ensure security in coding. They also found that these organizations had informal security best practices. Developers were expected to follow the best practices without any proper policies or guidelines.

In the same study highlighted, small organizations did not consider security as a dimension of the development process. Developers had no obligation to follow any secure coding practices.

6.1 Security Practices Adoption Framework

A structured approach to incorporating security practises into the software development life cycle (SDLC) in order to improve the overall security of software applications is referred to as the Security Practices Adoption Framework in software development. The framework outlines a series of steps that organisations can take to incorporate security practises into their existing SDLC processes. Steps may include:

- Assessment: Determining areas for improvement in the organization's security posture and assessing its current security posture.
- Planning: Creating a strategy to allocate resources and incorporate security practises into the SDLC.
- Training: Learning secure coding techniques, threat modelling, and security testing to software developers and other pertinent staff.
- Implementation: As much automation as possible while incorporating security procedures into the SDLC.
- Monitoring: Constantly keeping an eye on the software applications' security posture to make sure that security best practices are in place.
- Maintenance:Updating and maintaining security procedures on a regular basis, and adding new security tools and techniques as necessary, constitute maintenance.

By adhering to this framework, organiozations can implement security procedures in a structured and consistent manner, lowering the risk of security flaws and enhancing the security of their operations as a whole. They can also use the Security Practices Adoption Framework as a framework or methodology to develop and maintain a strong information security programme. It outlines a collection of best practises for protecting data, networks, and information systems. A variety of security-related topics are covered by the framework, including risk management, access control, incident response, and network security. It also offers instructions on how to successfully adopt and incorporate these practises into an organization's operations. The framework aims to assist organisations in creating a strong security posture, safeguarding against cyber threats, and adhering to pertinent laws and standards.To investigate security practices further, we need to understand the embracing of security practices by developers. The framework mocks up developers' preferences according to their perceptions of the benefits of security tools as illustrated in Figure. 1. Figure 1 explains about the Model of Security Practices Adoption Framework which comproses of Task, Action and Sanction as the main Component. It also included a manager who fully observes the implementation of practices and sanction mechanism that imposes a rule to use the security practices [30]. A sanction in this model can be positive or negative. It exhibits admonishment or reward. Considering the developers' adoption environment, violation of a rule is a negative sanction. When an individual is violating a rule, it recognizes as an individual sanction. However, when a sanction is referring to a group of individuals (or a subset of the group), it recognizes as group sanction [30].

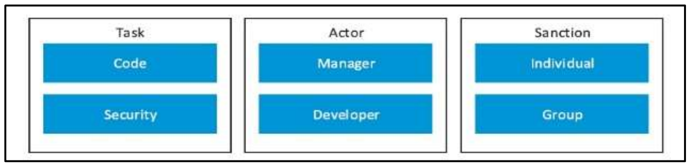

Figure 1: Model of Security Practices Adoption Framework [30]

The model reproduces the dimension of the interaction between a manager and developers in a time-critical project [30]. A manager is an agent responsible to assign tasks and enforce sanctions. A task in the model is referring to one of the three states - Not Coded, Coded, and Tested. Each agent (developer) has four attributes, which are Tasks, Coding and Security Skills, Developer health, and Preference. The tasks are the works allocated by the manager to a developer. Coding and Security are referred to as a developer skills level and tools usage. Developer health is referring to the completion of the task allocated to an agent. Preference is the possibility that an agent will select to do an action [30].

7. CONCLUSION

In this study, related articles on software development models are discussed, including the traditional model to the newer model called the hybrid model. The discussion discusses the strengths and weaknesses of each model and the rationale for combining the model with another model that meets the needs of a software project, resources and project scope. This study also takes seriously the software security practises of software teams and the policies that organisations use to ensure the security of a software system at every stage of development, from design to maintenance, such as OWASP and BSIMM. Not to be forgotten is the human factor in a software project. To create trustworthy programmes, programmers must have certain skills and knowledge, including tools. However, this study only focuses on the findings from the existing research in line with the research objective. In the near future, this research will explore the research gaps that emerge from the review conducted.

ACKNOWLEDGEMENTS:

This work is supported by the Ministry of Education (MOE) through Fundamental Research Grant Scheme (FRGS/1/2019/ICT01/UIAM/03/2).

REFERENCES:

[1] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring Software Security Approaches in Software Development Lifecycle: A Systematic Mapping Study," *Comput. Stand. Interfaces*, vol. 50, no. October 2016, pp. 107–115, 2017.

[2] M. U. A. Khan and M. Zulkernine, "Quantifying Security in Secure Software Development Phases," Proc. Int. Comput. Softw. Appl. Conf., pp. 955–960, 2008.

[3] K. Beck et al., "Manifesto for Agile Software Development," The Agile Alliance, 2001. [Online]. Available: http://agilemanifesto.org/.

[4] D. Ionita, C. Van Der Velden, H. K. Ikkink, E. Neven, M. Daneva, and M. Kuipers, "Towards Risk-Driven Security Requirements Management in Agile Software Development," Inf. Syst. Eng. Responsible Inf. Syst. CAiSE 2019, vol. 350, pp. 133–144, 2019.

[5] W. Laurie, "Secure Software Lifecycle - Knowledge Area," Natl. Cyber Security. Cent., no. 1, pp. 1–33, 2019.

[6] H. Assal and S. Chiasson, "'Think Secure from The Beginning': A Survey with Software Developers," CHI Conf. Hum. Factors Comput. Syst. Proc. (CHI 2019), vol. May 4–9, pp. 1–13, 2019.

[7] H. D. Benington, "Production of Large Computer Programs," Ann. Hist. Comput., vol. 5, no. 4, pp. 350–361, 1983.

[8] N. B. Ruparelia, "Software Development Lifecycle Models," ACM SIGSOFT Softw. Eng. Notes, vol. 35, no. 3, pp. 8–13, 2010.

[9] I. Sommerville, Software Engineering, 10th ed. Essex, England: Pearson Education Limited, 2015.

[10] M. A. Yasvi, K. S. Yadav, and Shubhika, "Review On Extreme Programming-XP," in International Conference on Robotics, Smart Technology and Electronics Engineering, At Delhi, 2019.

[11] I. Ghani, A. F. B. Arbain, Z. Azham, N. I. Yasin, and S. R. Jeong, "Integrating Security into Agile Models: Scrum, Feature-Driven Development (FDD), and eXtreme Programming (XP)," in Handbook of Research on Emerging Advancements and Technologies in Software Engineering, 1 edition., Hershey, USA: IGI Global, 2014, pp. 293–308.

[12] M. Kuhrmann et al., "Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond," ACM Int. Conf. Proceeding Ser., pp. 30–39, 2017.

[13] M. Kuhrmann et al., "Hybrid Software Development Approaches in Practice: A European Perspective," IEEE Softw., Jan. 2018.

[14] J. Klünder et al., "Catching up with Method and Process Practice: An Industry-Informed Baseline for Researchers," Proc. Int. Conf. Softw. Eng., no. ICSE-SEIP, May 2019.

[15] J. Klünder et al., "HELENA Study: Reasons for Combining Agile and Traditional Software Development Approaches in German Companies," Springer Int. Publ., vol. 10611, pp. 428–434, 2017.

[16] P. Tell et al., "What are Hybrid Development Methods Made Of? An Evidence-Based Characterization," in 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP), 2019, pp. 105–114.

[17] T. Hayata and J. Han, "A Hybrid Model for IT Project with Scrum," in Service Operations, Logistics, and Informatics (SOLI), 2011, pp. 285–290.

[18] D. Greer and G. Ruhe, "Software Release Planning: An Evolutionary and Iterative Approach," Inf. Softw. Technol., vol. 46, no. 4, pp. 243–253, Mar. 2004.

[19] M. Howard and S. Lipner, The Security Development Lifecycle. Washington, USA: Microsoft Press, 2006.

[20] Microsoft Corporation, "Microsoft Security Development Lifecycle," Microsoft Corporation, 2019. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl. [Accessed: 28-Oct-2019].

[21] OWASP, "OWASPTM Foundation," OWASP, 2019. [Online]. Available:

https://www.owasp.org/index.php/Main_Page. [Accessed: 30-Oct-2019].
[22] OWASP, "OWASP SAMM Project," OWASP, 2019. [Online]. Available: https://www.owasp.org/index.php/OWASP_SAMM_Project. [Accessed: 30-Oct-2019].
[23] OWASP, "OWASP Security Knowledge Framework," OWASP, 2019. [Online]. Available: https://www.owasp.org/index.php/OWASP_Security_Knowledge_Framework. [Accessed: 30-Oct-2019].
[24] BSIMM, "Building Security In Maturity Model," BSIMM, 2019. [Online]. Available: https://www.bsimm.com/. [Accessed: 29-Oct-2019].
[25] S. L. Kanniah and M. N. Mahrin, "A Review on Factors Influencing Implementation of Secure Software Development Practices," Int. J. Comput. Syst. Eng., vol. 10, no. 8, pp. 3032–3039, Sep. 2016.
[26] A. Meyer, L. Barton, G. Murphy, T. Zimmermann, and T. Fritz, "The Work Life of Developers: Activities, Switches and Perceived Productivity," IEEE Trans. Softw. Eng., vol. PP, p. 1, Jan. 2017.
[27] S. Xiao, J. Witschey, and E. Murphy-hill, "Social Influences on Secure Development Tool Adoption : Why Security Tools Spread," Proc. 17th ACM Conf. Comput. Support. Coop. Work Soc. Comput., no. CSCW '14, pp. 1095–1106, 2014.
[28] D. Oliveira, M. Rosenthal, N. Morin, K. C. Yeh, J. Cappos, and Y. Zhuang, "Its The Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developers Blind Spots," ACM Int. Conf. Proceeding Ser., vol. 2014-Decem, no. December, pp. 296–305, 2014.
[29] D. Baca, K. Petersen, B. Carlsson, and L. Lundberg, "Static Code Analysis to Detect Software Security Vulnerabilities - Does Experience Matter?," in 2009 International Conference on Availability, Reliability, and Security, 2009, pp. 804–810.
[30] S. Al-Amin, N. Ajmeri, H. Du, E. Z. Berglund, and M. P. Singh, "Toward Effective Adoption of Secure Software Development Practices," Simul. Model. Pract. Theory, vol. 85, pp. 33–46, 2018.
[31] F. D. Davis, "A Technology Acceptance Model For Empirically Testing New End-User Information Systems: Theory And Results," MA Inst. Technol., 1985.
[32] I. Ajzen and M. Fishbein, Understanding Attitudes and Predicting Social Behavior. Englewood Cliffs, N.J.: Prentice-Hall, 1980.
[33] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," MIS Q., vol. 13, no. 3, pp. 319–340, 1989.
[34] V. Venkatesh and F. D. Davis, "A Model of the Antecedents of Perceived Ease of Use: Development and Test*," Decis. Sci., vol. 27, no. 3, pp. 451–481, Sep. 1996.
[35] J. Xie, H. R. Lipford, and B. Chu, "Why Do Programmers Make Security Errors?," Proc. - 2011 IEEE Symp. Vis. Lang. Hum. Centric Comput. VL/HCC 2011, pp. 161–164, 2011.
[36] B. Naqvi and A. Seffah, "Interdependencies, Conflicts and Trade-Offs Between Security and Usability: Why and How Should We Engineer Them? BT - HCI for Cybersecurity, Privacy and Trust," 2019, pp. 314–324.