Effect of SDLC Models on The Perception of SSDLC Innovation Characteristics and SSDLC Adoption Intention

Wisdom Umeugo *wumeugo@gmail.com*
Ph.D. University of the Cumberlands
Independent Researcher
Ottawa, Canada

Kimberly Lowrey *kimberly.lowrey@ucumberlands.edu*
School of Computer and Information Sciences
University of the Cumberlands,
Kentucky, USA

Shardul Y. Pandya *shardul.pandya@ucumberlands.edu*
School of Computer and Information Sciences
University of the Cumberlands,
Kentucky, USA

Abstract

Software security remains an important issue. Security must be prioritized as a functional requirement to build secure software. Security must also be incorporated in every stage of the SDLC by practicing a secure SDLC (SSDLC). There are various SDLC models, each with emphasized priorities, strengths, and weaknesses. Increasing the security of more published software requires that SMEs, the majority of software publishers, adopt and practice the SSDLC. In promoting the SSDLC, there is a need to know if efforts should be adapted to the various SDLC models. This study empirically examined the effect of SDLC models on the innovation characteristics of the SSDLC derived from the Diffusion of innovation theory and the intention to adopt the SSDLC. A sample of software security managers of software SMEs in the United States was surveyed for the SDLC model used, their perception of the relative advantage, trialability, observability, complexity, and compatibility of the SSDLC, and intention to adopt the SSDLC. A Kruskal-Wallis test performed on the data showed no statistically significant differences between SDLC model groups for relative advantage, compatibility, trialability, observability, complexity, and intention to adopt the SSDLC. Results also indicated that SME Software security managers, on average, would be inclined to adopt the SSDLC if given the impetus. SSDLC adoption efforts can be mostly uniformly applied across the SDLC models. Software security policymakers may find the results of this study useful for SSDLC adoption policy formulation.

Keywords: Software Security, SSDLC, Secure Software, Diffusion of Innovation, Adoption.

1. INTRODUCTION

Software security is important and remains a recurring issue. New software vulnerabilities are reported daily in the National vulnerabilities database. Software's complexity necessitates engineering, which, in turn, requires a systematic approach to ensure that the software is successfully built according to requirements (Almazaydeh et al., 2022). This systematic approach to software development is termed the Software Development Lifecycle (SDLC) (Almazaydeh et al., 2022). The SDLC consists of various activities and processes grouped into phases from planning to development and deployment (Ragunath et al., 2010). The SDLC is practically applied as various models determined by the project's needs (Ragunath et al., 2010). Popular SDLC models include the Waterfall model, V-Model, Spiral model, incremental model, iterative

model, agile model, and prototyping model (Atawneh, 2019; Pressman & Maxim, 2014; Ragunath et al., 2010; Ruparelia, 2010). The SDLC models differ in their advantages, disadvantages, and emphasized values, such as planning, risk management, testing, flexibility, rapid development, and feedback (Kute & Thorat, 2014; Ragunath et al., 2010).

For software to be secure, security must be incorporated in all stages of its lifecycle (Al-Matouqet al., 2020; Khan et al., 2021). This implies that a security-infused SDLC, termed secure software development lifecycle (SSDLC), must be practiced. Integrating security practices into various SDLC models comes with challenges due to each SDLC model's inherent strengths, weaknesses, and priorities. SDLC models that focus on rapid development, usability, and shorter time-to-market, such as the agile model, do not emphasize security practices (Abdulrazeg et al., 2014; Boehm, 2002; McCaffery et al., 2018). Plan-driven development models such as the waterfall model and V-Model are better suited for developing high-assurance software and, therefore, are easier to incorporate security practices in all stages (Boehm, 2002; McCaffery et al., 2018). Therefore, efforts to increase SSDLC practice among software small and medium enterprises (SMEs), which are the majority of software publishers, may need to be tailored to the various SDLC models. This study investigated the effect of practiced SDLC models on the perception of the innovation characteristics of SSDLC and the intention to adopt SSDLC. The SSDLC innovation characteristics analyzed were those provided by Rogers' (2003) Diffusion of innovation theory (DOI). DOI is widely used to explain the adoption of new ideas, tools, and innovations in information technology and information security. The results of this study inform software security policymakers to help tailor software security adoption policies. The research also fills the existing knowledge gap on the effects of SDLC models on the perception of innovation characteristics and adoption intention of the SSDLC.

2. BACKGROUND AND HYPOTHESIS DEVELOPMENT

2.1 SDLC Models

The SDLC provides the order and cycle of activities or software lifecycle processes used to successfully build software according to requirements within pre-specified constraints like resources, timeframes, and costs (Acharya & Sahu, 2020; Adanna & Nonyelum, 2020). The SDLC generally involves a series of linear processes concerned with gathering the requirements, designing and developing the software, and testing, deploying, and maintaining the developed software (Arrey, 2019; Salve et al., 2018). The standard SDLC is practically applied to software development using multiple models that best suit the project's needs, constraints, and developers' choices (Acharya & Sahu, 2020). Historically, the concept of the SDLC was introduced in the 1960s when the development of large-scale business systems dominated the software industry and has since evolved from the need for structure and sequence to meet the demand for rapid feedback (Olorunshola & Ogwueleka, 2022; Ranawana & Karunananda, 2021). SDLC models followed the evolution of the software industry's needs, evolving from the sequential and rigidly structured waterfall model through the iterative model, the spiral model, the unified process model, and the widely used agile models (Ranawana & Karunananda, 2021). Boehm (2002) characterized this evolution as a shift from plan-driven development focused on high assurance to agile methods focused on rapid value. Pressman and Maxim (2014) characterized the SDLC models based on their order of activity or process flow into (1) linear or sequential, where each phase is performed after the previous phase has concluded; (2) evolutionary, where phases are performed circularly; (3) iterative, where phases are repeated before proceeding to the next phase; (4) parallel where phases are performed in parallel. Ragunath et al. (2010) classified the SDLC models into Linear models, evolutionary, formal systems development, agile methods, and reuse-based development. According to Ragunath et al. (2010), linear models such as the waterfall and V-Model linearly execute SDLC phases. Evolutionary models such as the spiral, incremental, and prototype models interleave the requirements and development phases (Ragunath et al., 2010). Formal Systems development formally transforms and implements mathematical system models, while reuse-based systems assemble software from existing components and modules (Ragunath et al., 2010).

2.1.1 The Waterfall SDLC

The Waterfall SDLC model is a rigid, sequential, and linear approach to the SDLC mainly used when all requirements are well-known and understood (Salve et al., 2018). The waterfall model process flows linearly from requirements analysis through system design, implementation, testing, deployment, and maintenance phases (Acharya & Sahu, 2020). The waterfall model is depicted in Figure 1. Each subsequent phase of the waterfall model starts only when the preceding phase is concluded and reviewed (Salve et al., 2018; Shaikh & Abro, 2019; Stoica et al., 2013). The waterfall model emphasizes planning and documentation, which is time-consuming but helps eliminate design errors. The waterfall model is suited to short projects where the requirements are clearly specified, a familiar technology is used, and expertise to conclude the project is readily available, such as projects involving the development of database-backed software (Kute & Thorat, 2014; Stoica et al., 2013). The waterfall model is still used across government and commercial projects (Acharya & Sahu, 2020).

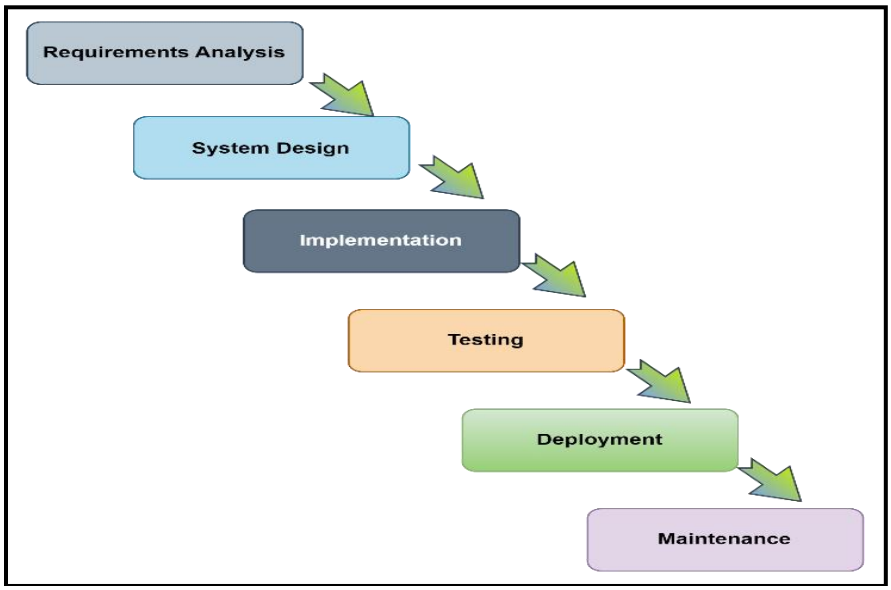

FIGURE 1: Waterfall SDLC model.

2.1.2 The V-Model SDLC

The V-Model SDLC model is a V-shaped variation of the waterfall model focused on quality assurance (Durmus et al., 2018; Pressman & Maxim, 2014). Verification and validation are emphasized in the V-Model (Acharya & Sahu, 2020). The V-Model is depicted in Figure 2. The left side of the V is the systems definition part composed of the linear sequential steps of planning, requirements analysis, system architecture determination, design, and implementation (Acharya & Sahu, 2020; Kargl et al., 2019). The right side is the systems verification part, consisting of unit testing, integration testing, system and acceptance testing, and deployment and maintenance (Acharya & Sahu, 2020; Kargl et al., 2019). The requirements, system architecture, and design phases on the left are each planned simultaneously with their equivalent step on the right so that testing is incorporated from the early stages of the SDLC (Acharya & Sahu, 2020; Kargl et al., 2019). This is shown in Figure 2 by bi-directional arrows within the “V.” The V-Model is plan-driven and preferable to the waterfall model for complex, quality-focused, and time-consuming projects, as in the automotive industry and medical software development projects (Akinsola et al., 2020; Kargl et al., 2019; McCaffery et al., 2018).

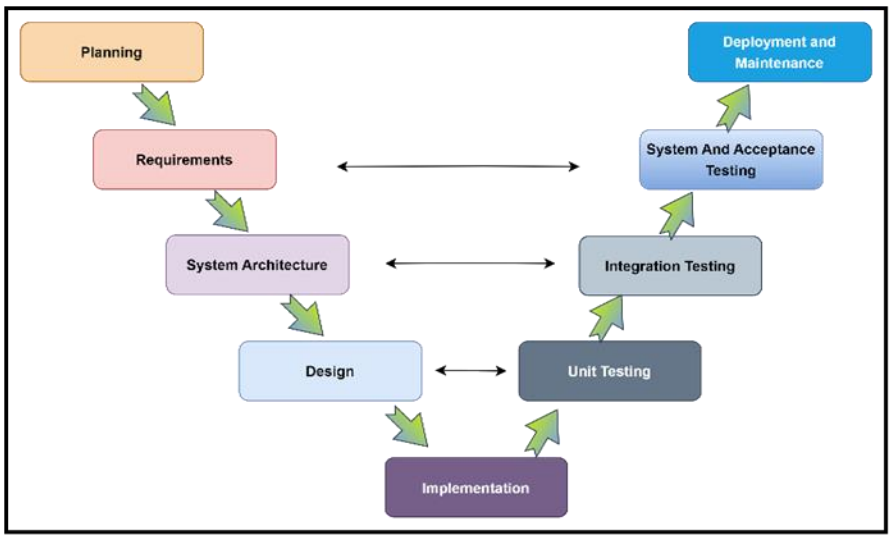

FIGURE 2: V-Model SDLC Model.

2.1.3 The Incremental SDLC Model

The incremental SDLC model splits the requirements into modules and implements them incrementally across multiple manageable waterfall-like development cycles (Saravanan et al., 2020; Stoica et al., 2013). The Incremental model is illustrated in Figure 3. A complete set of requirements is required to split all the requirements into module increments at the start of the incremental model (Stoica et al., 2013; Tsui et al., 2022). However, there is flexibility to incorporate minor changes due to feedback at the end of each increment (Stoica et al., 2013; Tsui et al., 2022). The first increment usually delivers a usable and releasable working minimal version of the software (Kute & Thorat, 2014; Stoica et al., 2013; Tsui et al., 2022). Each increment adds features and delivers a new version of usable working software (Stoica et al., 2013; Tsui et al., 2022). The cycle is repeated, incrementally adding the remaining modules until the software is completely developed (Saravanan et al., 2020; Stoica et al., 2013; Tsui et al., 2022). Three vertical dots depict this in Figure 3, implying increment cycle repetition until the last *n* increment cycle required to complete the project. The Incremental model is suitable for software projects with well-defined major requirements but requires some flexibility to evolve (Kute & Thorat, 2014; Stoica et al., 2013; Tsui et al., 2022). Projects that require using unfamiliar technology, have high inherent risks, and need to be quickly released are also suited to the incremental model (Kute & Thorat, 2014; Stoica et al., 2013).

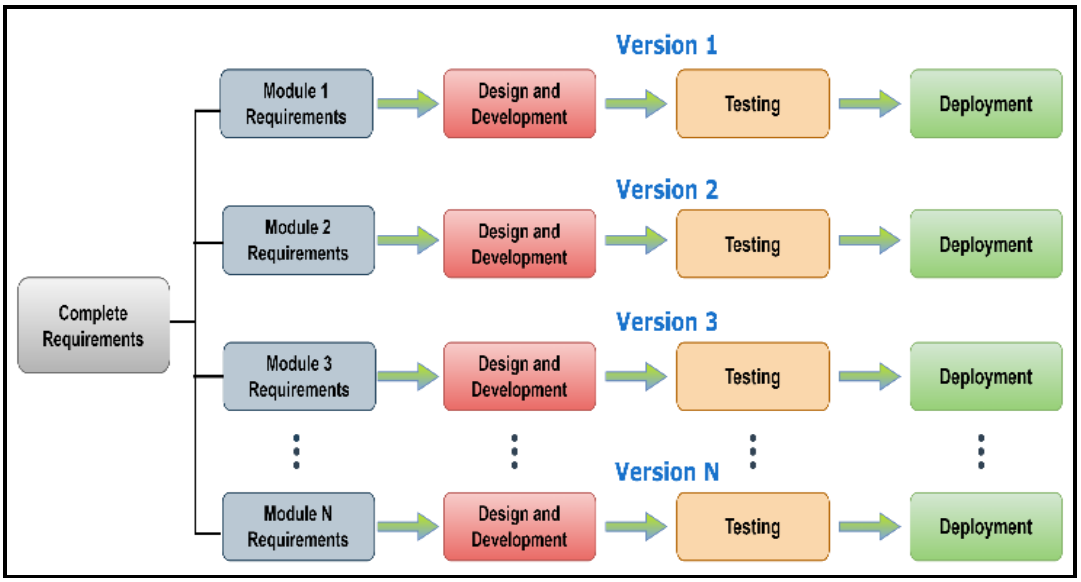

FIGURE 3: Incremental SDLC Model.

2.1.4 The Iterative SDLC Model

The Iterative SDLC model works similarly to the incremental model in that requirements are split and implemented over waterfall-like cycles (iteration or build) (Okesola et al., 2020). However, it differs in that all the core requirements are usually implemented in the first increment, while the advancement requirements are split into the remaining cycles (Okesola et al., 2020). The first iteration produces a working prototype with all the major requirements implemented. Therefore, subsequent iterations refine the first build until acceptance (Okesola et al., 2020). The iterative model is depicted in Figure 4. The product of each iteration of the iterative model is called a build (Olorunshola & Ogwueleka, 2022). The Iterative model only requires the major requirements to be specified initially, allowing refinements through minor requirements (Okesola et al., 2020). The iterative model is suited to the same types of software projects as the incremental model, except for those that require an early release to the market (Okesola et al., 2020).

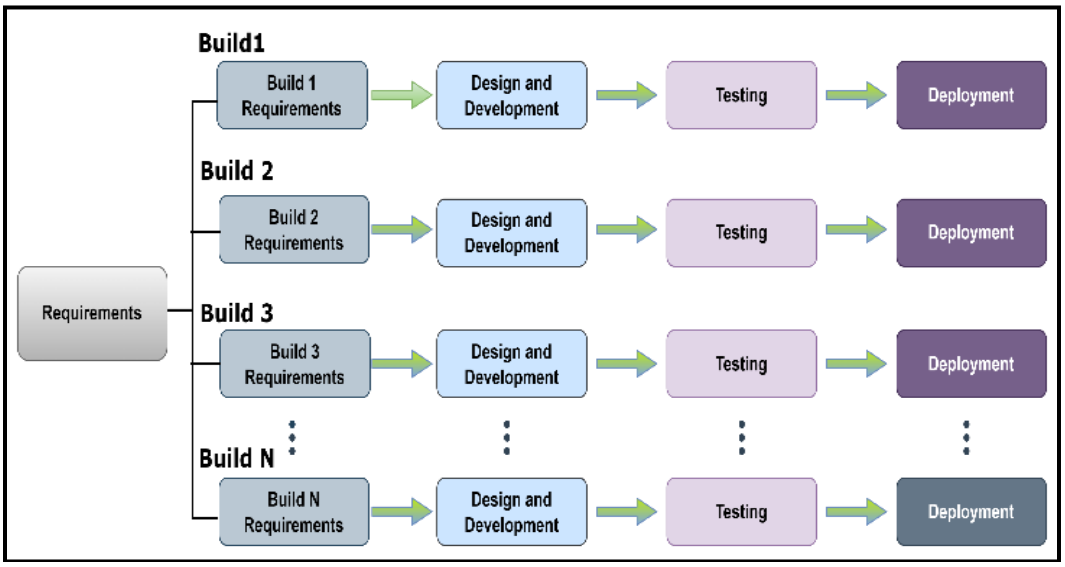

FIGURE 4: Iterative SDLC Model.

2.1.5 The Spiral SDLC Model

The Spiral SDLC model incorporates prototyping and risk management in an evolutionary spiral (Kute & Thorat, 2014). Prototyping focuses on rapidly developing a working set of features for feedback (Pressman & Maxim, 2014). Prototypes can typically evolve into the final software or be rejected and discarded (Pressman & Maxim, 2014). The Spiral model is illustrated in Figure 5. In the Spiral model, early iterations produce non-operational prototypes that may be used for demonstration and risk analysis (prototype one and prototype two in Figure 5). However, later spirals produce working software prototypes (Kute & Thorat, 2014). Each spiral consists of four phases depicted by quadrants in Figure 5: (a) Planning which involves requirements gathering and setting objectives; (b) Risk analysis, where the software's risks are analyzed, and a prototype is created; (c) Development: the prototyped software feature is developed using waterfall processes; (d) Evaluation: the resulting features are evaluated, and feedback is used to plan the next iteration of the spiral (Salve et al., 2018). The spiral model is preferred when core requirements are known, complex, and expected to evolve (Kute & Thorat, 2014). The spiral model is also suitable for projects with high costs, complexity, and risks, as in large and complex software projects like projects building new product lines (Kute & Thorat, 2014). The spiral model is used to develop U.S. military combat software (Salve et al., 2018).

The Spiral model has noted strengths. The spiral model adheres to the waterfall model during the development phase of the spiral, so it shares the advantages of having robust documentation and planning (Salve et al., 2018). The spiral model emphasizes attention to risk management, enhancing software quality (Salve et al., 2018). Building software in spirals, like the incremental model, enables software to be produced early that can be used to receive feedback (Salve et al., 2018).

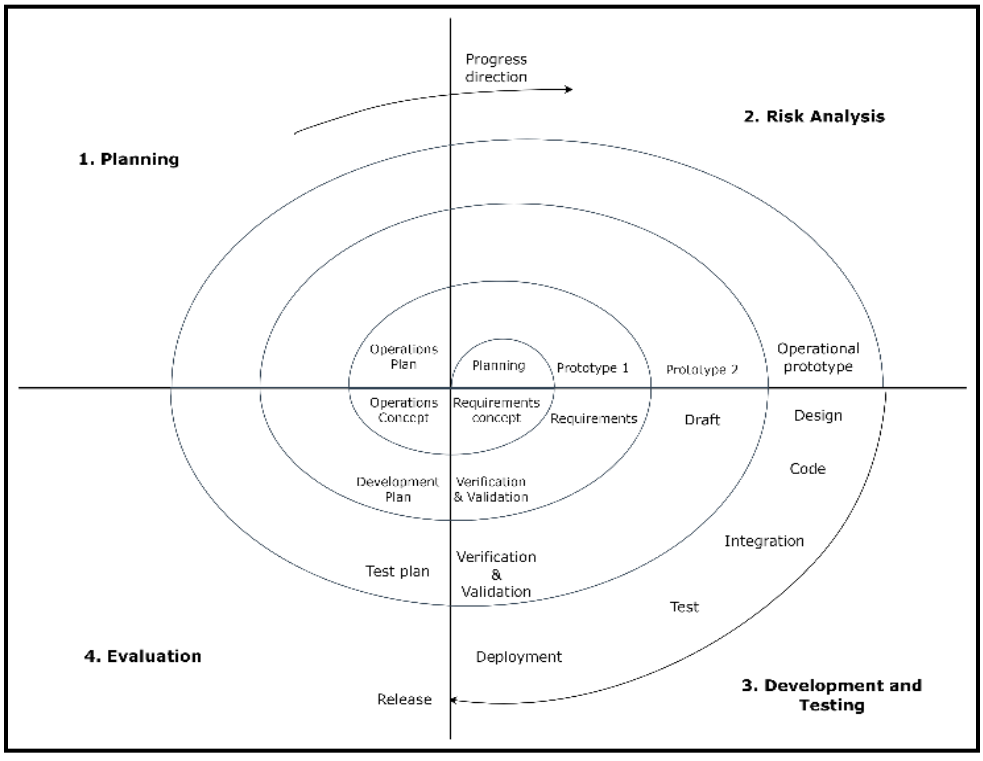

FIGURE 5: Spiral SDLC Model.

2.1.6 The Agile SDLC Model

The Agile SDLC model arose from the pervasiveness of change in software projects, the pressure to deliver, and the need to accommodate rapid changes (Pressman & Maxim, 2014;

Ranawana & Karunananda, 2021). The agile software development methodology professes twelve principles that favor effective communication, adaptation to changes, rapid production of working software, and self-organizing software teams (Mergel et al., 2020; Pressman & Maxim, 2014). This contrasts with traditional software development methods focusing on tools, documentation, planning, and contracts (Leau et al., 2012). Agile SDLC methods tend to avoid 'up-front' requirement gathering, preferring to promote customer collaboration and performing frequent demonstrations and releases of the software (Leau et al., 2012). Figure 6 depicts the agile development model. Figure 6 shows that multiple iterations are used to build the software iteratively or incrementally until completion. There is also an emphasis on getting feedback from the customer, which is then used to plan the next iteration. The agile method is modeled on incremental and iterative models depending on the development goals (Saravanan et al., 2020).

Various agile development models have been described in academic literature. These include Extreme programming (XP), Joint Application Development (JAD), Lean Development (LD), Dynamic Systems Development Method (DSDM), Agile Unified Process (AUP), Scrum, Crystal Method, Test Driven Development (TDD), and Feature-driven Development (FDD) (Al-Saqqaet al., 2020; Atawneh, 2019; Ibrahim et al., 2020; Pressman & Maxim, 2014; Ruparelia, 2010; Stoica et al., 2013). Agile methodology has become trendy in the software development industry primarily due to its suitability for small teams and small projects in small organizations and its focus on producing progressive results (Khalid et al., 2022). The agile model is particularly suited to small or medium-sized feature-driven projects where requirements are scarce upfront and are expected to change frequently (Olorunshola & Ogwueleka, 2022).

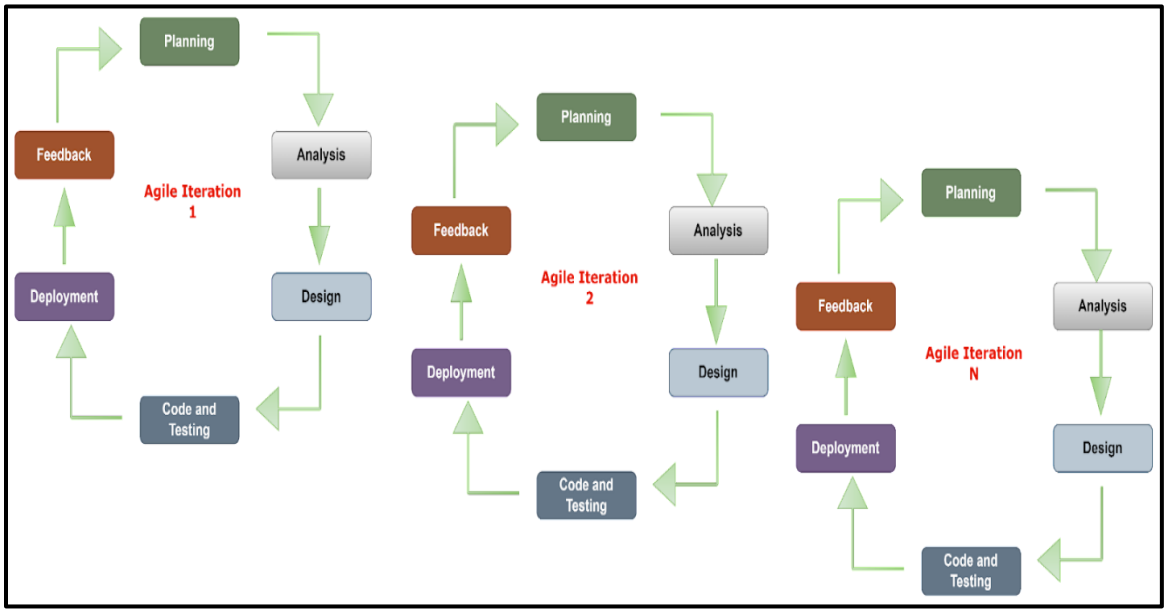

FIGURE 6: Agile SDLC Model.

2.2 SSDLC

The SSDLC is the SDLC infused with security activities at every phase (Alenezi & Almuairfi, 2019; Tudela et al., 2020). In the SSDLC, security requirements are gathered and included in requirements engineering during the requirements phase. Threat modeling, risk mitigation, security design, and security control selection are performed in the architecture design phase (Ransome & Misra, 2021; Ruggieri et al., 2019). In the development phase, security is ensured in the implementation by practicing secure and defensive coding, using secure programming languages and modules, performing peer code reviews, and static application testing (Alenezi & Almuairfi, 2019; Gasibaet al., 2020; Paul, 2013). The testing phase of the SSDLC includes various security testing such as dynamic application security test (DAST), fuzz testing,

vulnerability scanning, and penetration testing (Alenezi & Almuairfi, 2020; Paul, 2013; Ransome & Misra, 2021). Security activities in the operations and maintenance phase typically focus on issuing pre-release final security clearance, third-party security testing and certification, deployment environment hardening, and vulnerability management (Paul, 2013; Ransome & Misra, 2021).

2.3 SSDLC Innovation Characteristics

Rogers' (2003) diffusion of innovation (DOI) theory is the most widely used theory to explain information technology innovation adoption. DOI posits that five innovation characteristics, relative advantage, trialability, observability, complexity, and compatibility, impact its diffusion. The five DOI variables measured in this study are SME software security managers' perceptions of the five SSDLC innovation characteristics. Table 1 shows the definitions of the five innovation characteristics according to Rogers (2003).

Characteristic	Definition
Relative Advantage	"The degree to which an innovation is perceived as being better than the idea it supersedes"
Observability	"The degree to which the results of an innovation are visible to others"
Trialability	"The degree to which an innovation may be experimented with on a limited basis"
Complexity	"The degree to which an innovation is perceived as relatively difficult to understand and use."
Compatibility	"The degree to which an innovation is perceived as consistent with the existing values, past experiences, and needs of potential adopters"

TABLE 1: SSDLC Innovation Characteristics

According to DOI, among the five innovation characteristics, only complexity is expected to influence the innovation's adoption negatively. The adoption tendency of an information security innovation decreases as the complexity increases (Hameed & Arachchilage, 2020). The following five hypotheses were proposed to assess the effect of the SDLC model on SME software security managers' perception of the SSDLC's innovation characteristics.

***H1*:** There is a statistically significant difference in SME software security managers' perception of the relative advantage of SSDLC based on their practiced SDLC model.

***H2*:** There is a statistically significant difference in SME software security managers' perception of the compatibility of SSDLC based on their practiced SDLC model.

***H3*:** There is a statistically significant difference in SME software security managers' perception of the trialability of SSDLC based on their practiced SDLC model.

***H4*:** There is a statistically significant difference in SME software security managers' perception of the observability of SSDLC based on their practiced SDLC model.

***H5*:** There is a statistically significant difference in SME software security managers' perception of the complexity of SSDLC based on their practiced SDLC model.

2.4 SSDLC Adoption Intention

SSDLC adoption intention is the last assessed dependent variable. SSDLC adoption intention refers to the disposition towards adopting the SSDLC presently or shortly. SDLC models have different priorities and arrangements of lifecycle processes and activities that may pose challenges to integrating security into all its stages. Therefore, the intention to adopt the SSDLC

is expected to differ between practiced SDLC significantly. The sixth hypothesis is proposed to explore the effect of the SDLC model on SSDLC adoption intention.

H6: There is a statistically significant difference in SMEs' intention to adopt the SSDLC based on their practiced SDLC model.

3. RESEARCH METHOD

An online survey hosted on Pollfish was administered to a random sample from a population of software security managers and decision-makers of SMEs based in the United States. A minimum sample size of 216 was calculated by power analysis using G*Power ANOVA fixed effects, omnibus, one-way test at 0.25 effect size, 0.05 error probability, 0.80 power, and six groups. The survey consisted of demographic questions and five-point Likert scale variable measurement questions. The Likert scale measurement used ranged from 1=strongly disagree to 5=strongly agree. The questions measuring the variables were adapted from AlBar and Hoque (2019). Pollfish audience service provided participant recruitment based on the inclusion criteria. The survey was closed when 230 valid responses were received. Data from the survey was downloaded and imported into Jamovi statistical software. One-way ANOVA and Kruskal-Wallis tests were conducted on the data to determine the differences in perceptions of SSDLC's relative advantage, compatibility, complexity, observability, trialability, and SSDLC adoption intention based on the practiced SDLC model.

4. RESULTS

A total of 230 valid responses were received. One hundred thirty-five males and 95 females participated in the study. Most of the participants were aged between 25 and 44 years old. The majority of positions held by participants were the chief technology officer and chief executive officer positions. The Incremental model was the most used SDLC model, making up 28% of responses. Table. 2 shows the study's participant demographics. Table 3 shows the descriptive statistics for the variables.

Demographic	Category	Frequency (n)	Percent (%)
Age	25 – 34	92	40.0
	35 – 44	99	43.0
	45 – 54	28	12.2
	54+	11	4.8
Gender	Female	95	41.3
	Male	135	58.7
Experience	Less than three years	33	16.5
	3 – 5 years	38	19
	6 – 10 years	61	30.5
	11 – 15 years	30	15.0
	16 – 20 years	14	7.0
	20+ years	24	12.0
Organizational role	Chief Information Officer (CIO)	25	10.9
	Chief Information Security Officer (CISO)	22	9.6
	Chief Operation Officer (COO)	10	4.3
	Chief Technology Officer (CTO)	42	18.3
	Engineering Manager	18	7.8
	Information Security Manager	20	8.7
	Other	8	3.5

	Owner or Chief Executive Officer (CEO)	32	13.9
	Product Manager	19	8.3
	Software security Architect	19	8.3
	Tech Lead	15	6.5
SDLC Model	Agile model	37	16.1
	Incremental model	64	27.8
	Iterative model	30	13.0
	Spiral model	39	17.0
	V-Model	38	16.5
	Waterfall	22	9.6

TABLE 2: Participant Demographics.

	N	Mean	Standard deviation	Shapiro-Wilk W	Shapiro-Wilk p
Relative advantage	230	3.53	0.885	0.955	< .001
Complexity	230	3.34	0.904	0.971	< .001
Compatibility	230	3.6	0.895	0.957	< .001
Trialability	230	3.58	0.865	0.957	< .001
Observability	230	3.57	0.852	0.964	< .001
Intention	230	3.67	0.867	0.952	< .001

TABLE 3: Variable Descriptives.

All the SDLC groups had mean group values above average for all the dependent variables, indicating the average responses were “neither agree nor disagree” and “agree.” Table. 4 shows the group descriptives.

Variable	SDLC	N	Mean	SD	SE
Relative advantage	Agile	37	3.55	0.89	0.1463
	Incremental	64	3.54	0.914	0.1142
	Iterative	30	3.61	0.867	0.1583
	Spiral	39	3.51	0.729	0.1167
	V-Model	38	3.42	1.033	0.1676
	Waterfall	22	3.56	0.875	0.1866
Compatibility	Agile	37	3.68	0.938	0.1542
	Incremental	64	3.61	0.875	0.1094
	Iterative	30	3.76	0.742	0.1355
	Spiral	39	3.48	0.864	0.1384
	V-Model	38	3.54	0.888	0.1441
	Waterfall	22	3.48	1.153	0.2459
Trialability	Agile	37	3.44	1.009	0.1659
	Incremental	64	3.65	0.765	0.0956
	Iterative	30	3.96	0.699	0.1276
	Spiral	39	3.48	0.798	0.1277

	V-Model	38	3.54	0.928	0.1506
	Waterfall	22	3.32	0.984	0.2098
Observability	Agile	37	3.52	0.931	0.1531
	Incremental	64	3.66	0.684	0.0855
	Iterative	30	3.57	1.018	0.1858
	Spiral	39	3.56	0.899	0.1439
	V-Model	38	3.45	0.846	0.1373
	Waterfall	22	3.58	0.904	0.1927
Complexity	Agile	37	3.39	0.918	0.1509
	Incremental	64	3.44	0.812	0.1015
	Iterative	30	3.47	0.985	0.1798
	Spiral	39	3.44	0.876	0.1403
	V-Model	38	3.06	0.869	0.141
	Waterfall	22	3.14	1.082	0.2307
Intention	Agile	37	3.42	1.127	0.1852
	Incremental	64	3.71	0.856	0.1069
	Iterative	30	3.8	0.791	0.1444
	Spiral	39	3.74	0.843	0.135
	V-Model	38	3.68	0.733	0.1189
	Waterfall	22	3.65	0.766	0.1634

TABLE 4: Group Descriptives.

As shown in Table 4, The Iterative model had the highest mean of the groups for the perception of relative advantage ($\bar{x}$=3.61), compatibility ($\bar{x}$=3.76), trialability ($\bar{x}$=3.96), complexity ($\bar{x}$=3.47), and intention ($\bar{x}$=3.80). The incremental model had the highest group mean ($\bar{x}$=3.71) for observability. The group mean response of the V-Model ($\bar{x}$=3.06) and Waterfall model ($\bar{x}$=3.14) for complexity was noticeably low, indicating that software security managers in SMEs practicing the V-Model and Waterfall model were, on average, undecided about how complex the SSDLC would be to practice. The SDLC group mean scores for complexity were, on average, the lowest among the variables.

4.1 ANOVA Assumptions Test

The data were tested for ANOVA assumptions of linearity and homogeneity of variance. The data failed the linearity test because all dependent variables had statistically significant Shapiro-wilk test results in Table 3. All dependent variables except intention passed Levene's test for homogeneity of variance with statistically insignificant p-values. Table. 5 shows the results of Levene's test.

Variable	Statistic	df	df2	p
Relative advantage	0.729	5	224	0.603
Compatibility	1.705	5	224	0.134
Trialability	1.709	5	224	0.134
Observability	2.027	5	224	0.076
Complexity	0.664	5	224	0.651

Intention	3.192	5	224	0.008

TABLE 5: Levene's Test Result.

4.2 Kruskal-Wallis Test

Kruskal-Wallis test, the non-parametric equivalent of ANOVA, was conducted because the data failed ANOVA assumptions. Table 6 shows the result of the Kruskal-Wallis test. There were no statistically significant differences between groups of SDLC models for relative advantage (χ2 (5) = 0. 335, p =. 997), complexity (χ2 (5) = 6.749, p =. 240), compatibility (χ2 (5) = 1.534, p =. 909), trialability (χ2 (5) = 8.214, p =0.145), observability ($\chi 2$ (5) =1.557, p =. 906), and SSDLC adoption intention ($\chi 2$ (5) = 2.950, p =. 708).

Variable	χ^2	df	p
Relative advantage	0.335	5	0.997
Complexity	6.749	5	0.24
Compatibility	1.534	5	0.909
Trialability	8.214	5	0.145
Observability	1.557	5	0.906
Intention	2.95	5	0.708

TABLE 6: Kruskals-Wallis Test Result.

4.3 Hypotheses Testing

All hypotheses were tested by looking at the p-value for each dependent variable in the Kruskal-Wallis test result. All variables had statistically insignificant values (p > 0.05) in the Kruskal-Wallis result. All the hypotheses positing the existence of a statistically significant effect of the SDLC model on the dependent variable were, therefore, unsupported. Table VII shows the summary of the hypotheses tests.

Hypothesis	Significance	Result
***H1*:**There is a statistically significant difference in SME software security managers' perception of the relative advantage of SSDLC based on their practiced SDLC model.	0.997	Unsupported
***H2*:**There is a statistically significant difference in SME software security managers' perception of the compatibility of SSDLC based on their practiced SDLC model.	0.909	Unsupported
***H3*:**There is a statistically significant difference in SME software security managers' perception of the trialability of SSDLC based on their practiced SDLC model.	0.145	Unsupported
***H4*:**There is a statistically significant difference in SME software security managers' perception of the observability of SSDLC based on their practiced SDLC model.	0.906	Unsupported
***H5*:**There is a statistically significant difference in SME software security managers' perception of the complexity of SSDLC based on their practiced SDLC model.	0. 240	Unsupported
***H6*:**There is a statistically significant difference in SMEs' intention to adopt the SSDLC based on their practiced SDLC model.	0.708	Unsupported

TABLE 7: Summary of Hypothesis Test.

5. DISCUSSION

The above-average group response mean for all factors indicates a neutral or positive view of the SSDLC characteristics, which could translate to a greater intention to adopt the SSDLC. The iterative model had the highest group mean for most variables, including adoption intention. This may indicate a higher predisposition to adopt the SSDLC by organizations that practice the Iterative model. However, the mean group score of complexity for the iterative model was above average, indicating that the complexity of the SSDLC may pose a challenge in the iterative model. Group mean scores for complexity should ideally be lower than average because complexity negatively correlates with adoption intention. Complexity had the lowest group mean scores of all the variables. The V-Model and Waterfall model had the lowest group means, indicating that incorporating the SSDLC into the V-Model and Waterfall model may be less challenging than other SDLC models. This may be due to the plan-based nature of the V-Model and Waterfall models.

Results showed no statistically significant differences between groups of SDLC model in the perception of SSDLC innovation characteristics. The statistical insignificance between SDLC models in the perception of compatibility is surprising. SSDLC's compatibility with practiced SDLC models should naturally differ and be an essential factor in software SME SSDLC adoption intention because the more compatible an innovation is with existing practices, the greater the tendency to adopt it. The intention to adopt the SSDLC did not statistically significantly differ between groups of SDLC models. This implies that the practiced SDLC model does not significantly influence the intention to adopt the SSDLC. SME software security managers, therefore, may be willing to incorporate the SSDLC into their software development practices if they are given the impetus to adopt the SSDLC.

6. CONCLUSION

Incorporating security into all stages of the SDLC or practicing the SSDLC is critical for enhanced security of released software. No statistically significant differences were found between SDLC groups consisting of the agile model, incremental model, iterative model, spiral model, V-model, and waterfall model on the perception of relative advantage, compatibility, trialability, observability, and complexity of the SSDLC and the intention to adopt the SSDLC. The above-average group means, and the uniformity in the perception of SSDLC innovation characteristics and intention to adopt the SSDLC implies that efforts to improve SSDLC adoption in software SMEs do not have to consider the practiced SDLC model seriously. The complexity of the SSDLC is still an important factor, as highlighted by its above-average group means for most of the SDLC models. Efforts should be made to simplify practicing the SSDLC in the agile, incremental, spiral, and iterative SSDLC models. Future research could qualitatively explore incorporating the SSDLC into these SDLC models, producing practical and easy-to-implement frameworks for SSDLC practice.

7. REFERENCES

Abdulrazeg, A. A., Norwawi, N. M., & Basir, N. (2014). Extending V-model practices to support SRE to build secure web application. *2014 International Conference on Advanced Computer Science and Information System*, pp. 213–218. https://doi.org/10.1109/ICACSIS.2014.7065838

Acharya, B., & Sahu, K. (2020). Software development life cycle models: A review paper. *International Journal of Advanced Research in Engineering and Technology (IJARET)*, *11*, 169–176.

Adanna, A. A., & Nonyelum, O. F. (2020). Criteria for choosing the right software development life cycle method for the success of software project. *IUP Journal of Information Technology*, *16*(2), 39–65.

Akinsola, J. E., Ogunbanwo, A. S., Okesola, O. J., Odun-Ayo, I. J., Ayegbusi, F. D., & Adebiyi, A. A. (2020). Comparative analysis of software development life cycle models (SDLC). In *Intelligent*

Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9 (pp. 310–322). Springer International Publishing.

AlBar, A. M., & Hoque, M. R. (2019). Factors affecting cloud ERP adoption in Saudi Arabia: An empirical study. *Information Development*, *35*(1), 150–164. https://doi.org/10.1177/0266666917735677

Alenezi, M., & Almuairfi, S. (2019). Security risks in the software development lifecycle. *International Journal of Recent Technology and Engineering*, *8*(3), 7048-7055.

Alenezi, M., & Almuairfi, S. (2020). Essential activities for secure software development. *International Journal of Software Engineering & Applications (IJSEA)*, *11*(2).

Al-Matouq, H., Mahmood, S., Alshayeb, M., & Niazi, M. (2020). A maturity model for secure software design: A multivocal study. *IEEE Access : Practical Innovations, Open Solutions*, *8*, 215758–215776. https://doi.org/10.1109/ACCESS.2020.3040220.

Almazaydeh, L., Alsafasfeh, M., Alsalameen, R., & Alsharari, S. (2022). Formalization of the prediction and ranking of software development life cycle models. *International Journal of Electrical and Computer Engineering (IJECE)*, *12*(1), 534. https://doi.org/10.11591/ijece.v12i1.pp534-540.

Al-Saqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile software development: methodologies and trends. *International Journal of Interactive Mobile Technologies (IJIM)*, *14*(11), 246. https://doi.org/10.3991/ijim.v14i11.13269.

Atawneh, S. (2019). The analysis of current state of agile software development. *Journal of Theoretical Applied Information Technology*, p. *97*.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64–69. https://doi.org/10.1109/2.976920.

Arrey, D. A. (2019). Exploring the integration of security into software development life cycle (SDLC) methodology (Doctoral dissertation, Colorado Technical University).

Durmus, M. S., Ustoglu, I., Tsarev, R. Y., & Börcsök, J. (2018). Enhanced V-Model. *Informatica*, *42*(4). https://doi.org/10.31449/inf.v42i4.2027.

Gasiba, T. E., Lechner, U., Pinto-Albuquerque, M., & Fernandez, D. M. (2020, December). Awareness of secure coding guidelines in the industry-A first data analysis. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)* (pp. 345–352). IEEE.

Hameed, M. A., & Arachchilage, N. A. G. (2020). A conceptual model for the organizational adoption of information system security innovations. In R. C. Joshi & B. B. Gupta (Eds.), *Security, privacy, and forensics issues in big data* (pp. 317–339). IGI Global. https://doi.org/10.4018/978-1-5225-9742-1.ch014.

Ibrahim, M., Aftab, S., Bakhtawar, B., Ahmad, M., Iqbal, A., Aziz, N., Javeid, M. S., & Ihnaini, B. N. (2020). Exploring the agile family: A survey. *IJCSNS*, *20*(10).

Kargl, F., Schmidt, R., Kung, A., & Bösch, C. (2019). A privacy-aware V-model for software development. *2019 IEEE Security and Privacy Workshops (SPW)*, 100.

Khalid, A., Butt, S. A., Jamal, T., & Gochhait, S. (2022). Agile Scrum Issues at Large-Scale Distributed Projects: Scrum Project Development At Large. In I. R. Management Association (Ed.), *Research anthology on agile software, software development, and testing* (pp. 388–398). IGI Global. https://doi.org/10.4018/978-1-6684-3702-5.ch019.

Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2021). Systematic mapping study on security approaches in secure software engineering. *IEEE Access : Practical Innovations, Open Solutions*, *9*, 19139–19160. https://doi.org/10.1109/ACCESS.2021.3052311.

Kute, S. S., & Thorat, S. D. (2014). A review on various software development life cycle (SDLC) models. *International Journal of Research in Computer and Communication Technology*, *3*(7), 778–779.

Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. *International Conference on Information and Network Technology*, *37*(1), 162.

McCaffery, F., Özcan-Top, Ö., Treacy, C., Paul, P., Loane, J., Crilly, J., & Mahon, A. M. (2018). A process framework combining safety and security in practice. In X. Larrucea, I. Santamaria, R. V. O'Connor, & R. Messnarz (Eds.), *Systems, Software and Services Process Improvement: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings* (Vol. 896, pp. 173–180). Springer International Publishing. https://doi.org/10.1007/978-3-319-97925-0_14.

Mergel, I., Ganapati, S., & Whitford, A. B. (2020). Agile: A new way of governing. *Public Administration Review*. https://doi.org/10.1111/puar.13202.

Okesola, O. J., Adebiyi, A. A., Owoade, A. A., Adeaga, O., Adeyemi, O., & Odun-Ayo, I. (2020). Software requirement in iterative SDLC model. In R. Silhavy (Ed.), *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1* (Vol. 1224, pp. 26–34). Springer International Publishing. https://doi.org/10.1007/978-3-030-51965-0_2.

Olorunshola, O. E., & Ogwueleka, F. N. (2022). Review of system development life cycle (SDLC) models for effective application delivery. In A. Joshi, M. Mahmud, R. G. Ragel, & N. V. Thakur (Eds.), *Information and communication technology for competitive strategies (ICTCS 2020) ICT: applications and social interfaces* (Vol. 191, pp. 281–289). Springer Singapore. https://doi.org/10.1007/978-981-16-0739-4_28.

Paul, M. (2013). *Official (ISC)2 Guide to the CSSLP CBK ((ISC)2 Press)* (2nd ed.). Auerbach Publications.

Pressman, R., & Maxim, B. (2014). *Software engineering: A practitioner's approach* (8th ed.). McGraw Hill.

Ragunath, P. K., Velmourougan, S., Davachelvan, P., Kayalvizhi, S., & Ravimohan, R. (2010). Evolving a new model (SDLC Model-2010) for software development life cycle (SDLC). International Journal of Computer Science and Network Security, 10(1), 112-119.

Ranawana, R., & Karunananda, A. S. (2021). An agile software development life cycle model for machine learning application development. *2021 5th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI)*, 1–6. https://doi.org/10.1109/SLAAI-ICAI54477.2021.9664736.

Ransome, J., & Misra, A. (2021). *Core software security* (1st ed.). Routledge.

Rogers, E. M. (2003). Diffusion of innovations. *NY: Simon and Schuster*, p. *576*.

Ruggieri, M., Hsu, T. T., & Ali, M. L. (2019, October). Security considerations for the development of secure software systems. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (pp. 1187–1193). IEEE.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8. https://doi.org/10.1145/1764810.1764814.

Salve, S. M., Samreen, S. N., & Khatri-Valmik, N. (2018). A Comparative Study on Software Development Life Cycle Models. *International Research Journal of Engineering and Technology (IRJET)*, *5*(2), 696–700.

Saravanan, T., Jha, S., Sabharwal, G., & Narayan, S. (2020). Comparative analysis of software life cycle models. *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pp. 906–909. https://doi.org/10.1109/ICACCCN51052.2020.9362931.

Shaikh, S., & Abro, S. (2019). Comparison of traditional and agile software development methodology: a short survey. *International Journal of Software Engineering and Computer Systems*, *5*(2), 1–14. https://doi.org/10.15282/ijsecs.5.2.2019.1.0057.

Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software development: agile vs. traditional. *Informatica Economica*, *17*(4/2013), 64–76. https://doi.org/10.12948/issn14531305/17.4.2013.06.

Tsui, F., Karam, O., & Bernal, B. (2022). *Essentials of software engineering*. Jones & Bartlett Learning.

Tudela, F. M., Higuera, J. R. B., Higuera, J. B., Montalvo, J. A. S., & Argyros, M. I. (2020). On combining static, dynamic and interactive analysis security testing tools to improve OWASP top ten security vulnerability detection in web applications. *Applied Sciences*, *10*(24), 9119.