IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 10, 2019, accepted January 8, 2020, date of publication February 3, 2020, date of current version February 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2971000

A Novel Lightweight Solo Software Development Methodology With Optimum Security Practices

SIBONILE MOYO AND ERNEST MNKANDLA
School of Computing, University of South Africa, Florida 119172, South Africa
Corresponding author: Sibonile Moyo (sibonile.moyo@nust.ac.zw)

ABSTRACT The diffusion of software into all areas of life and all forms of business, increases the demand for high-quality and secure software products. Software development methodologies are designed to improve the quality of software by incorporating practices that promote quality in the developed software. Software security is an important facet of software quality, particularly in this era, where most software is deployed for use over the Internet. Most research on developing high-quality and secure software is normally focused on teams at the expense of individual developers. In trying to fill this gap, in this paper we propose an agile secure-software development methodology. We design a methodology that promotes quality and security in the software products of solo developers. We integrate quality practices with lightweight security practices to produce agile secure software development practices. We draw quality practices from a solo software development framework designed in our previous study, while security practices are drawn from existing lightweight methodologies. We adapt Keramati and Mirian-Hosseinabadi's algorithm to integrate the two sets of practices, taking care to maintain an optimum degree of agility in the target methodology. We evaluate the utility of the resultant methodology through a case study. Results from the case study show that our proposed methodology can be used to build quality and secure software products without compromising the agility of the methodology.

INDEX TERMS Agility degree, quality practices, security practice, software development methodology, software product, software quality, software security, solo developer.

I. INTRODUCTION

Software security is an important characteristic of software quality, especially in this era, where most software applications are deployed over the Internet. In this regard, developing secure software has become a topical research area among a number of authors [1]–[4], particularly for those software applications that are designed to handle online transactions [5]–[8]. As most business applications become accessible online, security problems due to code vulnerability have increased, resulting in loss of personal and financial data to both individuals and organizations [2]. Web applications in particular, have been shown to be prone to SQL injection vulnerabilities, necessitating the need to design secure software development practices to counter such threats [8].

Mobile and web applications tend to be popular with freelance developers due to their small size. Their increased deployment in business has also seen an increase in the number of freelance developers in the software industry. Freelance developers are individuals with the sole responsibility of delivering high-quality software to a client or to a target market. In this study we also refer to these developers as solo developers.

A number of lightweight solo software development methods (SSDMs) [9]–[12] exist. Whereas these methods have been designed with the aim to produce quality software products, none of the methods found in the literature address the security aspect of the developed software. This was shown in our previous work in [13]. A mapping of the quality framework resulting from the meta-synthesis of the existing SSDMs against the ISO/IEC 25010 quality model defined in [14], showed that existing methodologies lack practices to support software security, among other quality characteristics. This model divides quality characteristics into high-level and low-level characteristics. Low-vel characteristics are what can be measured, and high-level characteristics are what is expected of the software. Table 1 summarizes the mapping of the quality practices against the quality characteristics.

The associate editor coordinating the review of this manuscript and approving it for publication was Hui Liu.

TABLE 1. Mapping quality practices against iso/iec 25010 quality characteristics.

Quality practices derived from existing SSDMS [13]	Quality characteristics defined in the ISO/IEC 25010 quality model [14]	
	Sub-characteristics	Characteristics
Creation of product backlog, requirements checklist, work break down structure	Functional completeness	Functional suitability
Development standards, automated code review, use of a dummy partner, test driven development	Functional correctness	
Creation of product backlog, product validation	Functional appropriateness	
Nil	Time behavior Resource utilization Capacity	Performance efficiency
Integration testing, system testing	Co-existence, Interoperability	Compatibility
Product validation, use of development standards, simple metaphors	Appropriateness Learnability Operability User error protection User interface aesthetics, Accessibility	Usability
Adoption of development standards, product validation	Maturity Availability Fault tolerance Recoverability	Reliability
Nil	Confidentiality Integrity Non-repudiation Accountability Authenticity	Security
Small user stories, unit testing, task independence Refactoring, simple designs, use of version control system	Modularity Testability Analyzability	Maintainability
Nil	Adaptability Install ability Replaceability	Portability

As shown in the table, creation of product backlog, use of a requirements checklist, and creation of a work-breakdown structure promote functional completeness, which result in product suitability. Similarly, use of development standards, automated code review, use of a dummy partner in reviewing code and test-driven development promote functional correctness. Functional correctness also contributes to functional suitability. The rest of the table can be interpreted in the same manner. Table 1 shows that some quality characteristics from the quality model are not supported by the framework. Characteristics not supported include performance efficiency, portability and security. The view of lack of secure practices in agile methods, is corroborated by several other authors [2], [3], [15].

Given this background, we aim to address this shortcoming of agile methods, particularly for the solo software development environment. In designing a software development methodology (SDM) that promotes software quality in a solo environment, we consider the issue of resource constraints, which is characteristic of this environment. A befitting design would therefore be that which embeds quality and security promoting practices into the SDM. Embedding secure software development practices into agile methods is however not an easy task [15]–[17]. Adding available security promoting practices to agile methods may compromise the agility of the resultant method if appropriate measures are not taken. There is therefore a need to artfully integrate security practices with agile quality practices without compromising the agility of

the resultant SDM. Agility is key for an SSDM as this would facilitate its uptake, especially by independent developers.

We tackle this challenge to integrate lightweight security practices with agile quality practices to build an agile solo software development methodology. In drawing the practices, we restrict our consideration to the solo development environment. We pose the following main question for the purpose:

How can existing secure software development practices be integrated with lightweight quality software development practices to build a secure solo software development methodology (Secure-SSDM), without compromising the resulting methodology's agility?

To help answer this main question, we further pose the following sub-questions:

Q1: What quality and lightweight security practices exist for agile software development?

Q2: How can the lightweight security practices and agile quality practices be integrated into a software development methodology without compromising the agility of the resultant practices?

Q3: How can the utility of the resulting methodology be evaluated?

In pursuing answers to the questions above, we first derive SSDM core quality practices and security practices that can be executed by an individual from the literature. Then we compute agility degrees for these independently. After that we formulate a compatibility matrix to ease the integration process. We then use a modified version of Keramati and Mirian-Hosseinabadi's algorithm [15] to integrate the security practices and the quality practices taking care to maintain the agility of the resultant practices. We conclude by evaluating the utility of the proposed Secure-SSDM through a case study. In the following sections of the paper we discuss this process of designing and evaluating the proposed Secure-SSDM. Section II discusses efforts made by other researchers to build quality and secure-lightweight methodologies. Section III discusses the methodology used to design the proposed Secure-SSDM and overviews the resulting methodology. Section IV evaluates the Secure-SSDM, and discusses threats to validity, while Section V presents the conclusion and recommendations for future work.

II. RELATED WORK

Solo software development has attracted the attention of a number of researchers. This stems from the increase in the number of software projects that can be undertaken by an individual working alone [12]. In a bid to improve the quality of software by these developers, research has sought to design lightweight methods that can be used by these individuals in building quality into their software products. Ramingwong, Ramingwong and Kusalaporn [12], scaled down Scrum practices to produce Solo-Scrum. In this scaled down version the developer takes on all the Scrum roles and designs the expected intermediate artefacts like the product backlog and sprint backlog. Since the daily meetings characteristic of the scrum team are not necessary in a solo environment, the developer holds meetings with the product owner at three-day intervals. To evaluate the utility of Solo-Scrum, a case study was used. In this case study, a solo developer applied the methodology in developing a web-based application. Perceptions of both the developer and project participants were collected for the evaluation purpose.

González-Sanabria, Morente-Molinera and Castro-Romero [18] designed DeSoftIn, a methodology for use by independent developers in an academic setting. The authors synthesized quality practices from identified agile methods for the purposes of guiding students working on individual projects. They incorporated a number of quality practices including a requirements checklist that is used to capture requirements. The checklist is color-coded during development to keep track of developer progress. At the implementation, stage, the developer tests the software products for compliance with quality and security standards. While the authors provide quality practices for improving the quality of the software products, they do not discuss security practices to be used to build security into the product. They however acknowledge the importance of security in the developed software.

Several researchers [15], [17], [19], [20] have tackled the problem of improving software quality through introducing traditional security practices into lightweight methods. The authors in [20] extend the agile development methodology (ADM) by incorporating security practices in the stages of the methodology. In the initiation phase developers and the users work together to identify software security risks. For each identified risk, the probability and impact of the risk is derived based on the user's perspective. This is used to rank the risks. Goals for mitigating the risks are then formulated and presented as security user stories. Users choose which threats to mitigate in a given iteration. On implementing the functional and security user stories, developers produce software to comply with security requirements of the user. This work is similar to ours in that we also incorporate security practices at each stage of the development process. It differs in that in their case the development environment is team-based and the authors use a risk-based approach for security analysis, while we do not. Further, to minimize costs in a solo development environment we do not perform the security assurance reassessment of implemented modules at the end of every iteration.

In [19] the authors consider the possibility of introducing traditional software development security practices into agile methods. The authors analyze the compatibility of the traditional security practices with agile practices. They conclude that compatible and independent security practices are readily integratable with existing agile methods. They recommend automation supported by knowledge management for partially automatable practices. For the mismatch practices, they suggest designing a new set of agile security practices or applying traditional security practices at least two times within the agile development process. We consider these authors as some form of feasibility study. It differs from

ours in that we propose a method that incorporates existing practices into the life cycle. The authors' work is important for our purposes in that it points out the feasibility of building a secure agile software development methodology.

Keramati and Mirian-Hosseinabadi [15] developed an algorithm that is used to identify security practices for the purposes of integrating these with agile practices. To maintain the agility of the resulting practices, the algorithm computes an agility degree for the identified security practice, after which the practice is integrated, if and only if, it meets a certain agility threshold. The team or organization wishing to introduce security practices into its agile development processes in this case determines the agility threshold. The team conducts a thorough analysis of its environment and its capabilities to handle security practices in order to come up with its agility threshold value. The use of a threshold value, while not applicable as is in this paper, can be adapted to put a lower limit on the agility degrees of practices so as to determine those practices that are integratable and those that are not. We therefore adopted these authors algorithm to integrate quality and security practices drawn from existing methodologies to produce secure agile practices.

Fisa-XP [17] is a framework designed through combining eXtreme Programming (XP) practices with secure development practices. The security practices are drawn from Open Web Application Security Project (OWASP)'s Comprehensive Lightweight Application Security Process (CLASP). The authors of Fisa-XP use a modified version of the algorithm in [15] to identify appropriate security practices from CLASP which they integrate with XP practices. They also developed Tisa-XP, an automated tool, which they use for the purposes of integrating the quality and security practices. This automated tool serves as a knowledge base to help ease the integration process. Tisa-XP also provides an inbuilt tutorial to assist developers with the integration. The utility of Tisa-XP was evaluated through a case study carried out in a software development organization. After using the tool, developers were asked to comment on its ease of use, time saving aspect of the tool and whether it was interesting to use. The responses were in support of the tool.

Case studies have been widely used in software engineering to demonstrate the usability of the resulting software development methodologies in a live setting. Besides the authors cited in [17] above, Hamid and Weber [21] used a case study carried out in an industry setting, to evaluate their model driven methodology's utility in developing secure software. The authors demonstrated the feasibility of their approach through its application in a metrology environment. Participants perception of the methodology were collected through a survey conducted at the end of the case study. An online questionnaire was used to collect these anonymously. Perceptions collected from the participants confirmed the model's ease of use and its applicability in the industry. In addition to using expert evaluation, Quiñones, Rusu and Rusu [22] used a case study to show the utility of their methodology in developing usability heuristics. The methodology was applied in various environments in different case studies for the purpose. Similarly, the perceptions of the participants in the different case studies were collected through a survey after the case study. While the responses from the participants indicated that the methodology was difficult, participants opined that it was usable in their environment and they indicated that they would use it in the future. These examples have shown the popularity of case studies in evaluating new methods.

Case studies are used normally to evaluate an artifact in its natural setting. They are appropriate in evaluation cases where the boundary between the environment and the artifact is not clear. This is true for software development methodologies where a number of variables have an impact on the success of the software project. Evaluating a software development methodology in its intended setting helps to determine the acceptability of the methodology by its intended audience. The Secure-SSDM is evaluated in an academic using a case study and perceptions of the student participants after using the methodology are collected through focus group discussions and document analysis. The collected data is analyzed qualitatively.

Most of the works we found in the literature, particularly on secure software development concentrate on teams as can been seen in the preceding paragraphs. Our work is unique in that we focus on developing a methodology targeted at individual software developers. We use the quality framework in [13] as a source for the core quality practices of the Secure-SSDM. The security practices in [23] are used as a source for security practices. We consider this pool of security practices to be consensus practices in the agile environment as they are derived through a systematic literature review. In integrating the identified lightweight security practices with the quality promoting practices, we identify only those security practices that can be performed by an individual and we systematically integrate these with the SSDM framework discussed in [13] to produce the proposed Secure-SSDM. We discuss the identification and integration process in the following methodology section.

III. METHODOLOGY

We use Design Science Research (DSR) [24] to iteratively design the proposed Secure-SSDM. We deem it applicable in this paper as we seek to design a satisficing solution to integrate security and quality practices for the purposes of embedding quality in the software development process. We adopt the DSR cycle of: (1) Problem identification; (2) Definition of solution objectives; (3) Design and development; (4) Solution demonstration; (5) Solution evaluation; and (6) Results communication. This paper focuses on the design, demonstration and evaluation of the Secure-SSDM. The problem is highlighted in our research question cited above, as how to embed security practices into agile quality practices without compromising the agility of the resultant methodology. The design process is summarized in Figure 1.

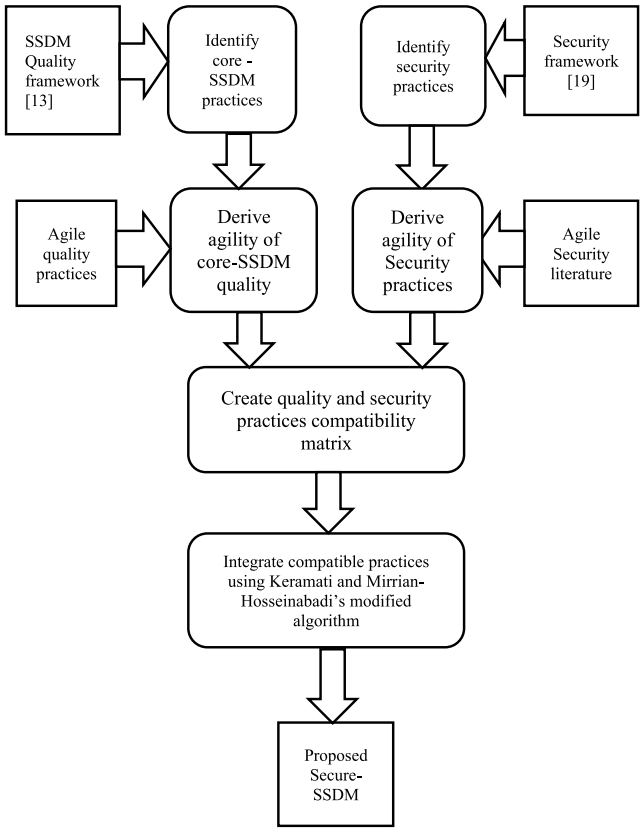

FIGURE 1. Methodology processes in designing the Secure-SSDM.

As Figure 1 shows, we derive the two different types of practices to be embedded in the Secure-SSDM from the literature. Core-SSDM quality practices are drawn from the SSDM quality framework discussed in [13] to create a quality practices list, while security practices are drawn from the security framework in [23]. The quality practices in the latter framework are drawn from established secure software development methods, hence are considered to be acceptable practices in secure software development. Sub-sections A to D elaborate the activities in the figure.

A. IDENTIFYING QUALITY AND SECURITY PRACTICES

We identify the core development practices of the Secure-SSDM through selecting those practices confirmed by two or more authors of the SSDMs participating in the meta-synthesis in [13]. This way, the Secure-SSDM is built on development practices generally accepted in the SSDM community [24]. The rest of the practices in the framework remain as non-core practices which are executed depending on the type of product under development. We also identify security practices from the framework in [23] to create a security practices list. We identify those security practices that can be executed by an individual, and place them in the related stage of the quality framework. Table 2 shows the list of quality and security practices drawn from the literature. They have been organized into development stages to facilitate the integration process. The list of quality and security practices provide us with the answer to sub-question Q1. Having identified the practices, we proceed to compute their degrees of agility independently.

TABLE 2. Quality and security practices.

SSDM Quality practices [13]	**Security practices** [23]
I. Management Buy-in and Standards Adoption	
-User identification & education -Standards (quality & development) adoption	-Security awareness training -Security analysis of user roles
II. Requirements Elicitation	
-Creation of user stories -Creation of use cases -Creation & prioritization of product backlog -Prototype development	-Creation of misuse cases -Misuse case detailing
III. Release and Sprint Planning	
-Prioritization of sprint tasks & sprint planning -Use of short sprints (1-2 weeks) -Design of acceptance tests	-Application of security design principles -Security test design
IV. Development with Review	
-Coding & code review (with dummy) -Version/change control tracking -Code refactoring	-Security coding - standard adherence -Source code security reviews
V. Sprint Review and Close	
-Product validation/Deliverables evaluation -Code integration & testing -Sprint & project progress review	-Security testing
VI. Evaluation	
-System testing -Task automation	-Security disclosure management -Review of security repository

B. COMPUTING THE AGILITY DEGREES OF PRACTICES

In determining the degrees of agility of these two types of practices, this paper adopts Qumer and Henderson-Sellers's definition of agility thus:

"Agility is a persistent behavior or ability of a sensitive entity that exhibits flexibility to accommodate expected or unexpected changes rapidly, follows the shortest time span, uses economical, simple and quality instruments in a dynamic environment and applies updated prior knowledge and experience to learn from the internal and external environment" [25].

These authors derive five features of agility from this definition. These are: flexibility, swiftness, leanness, responsiveness, and learning. We add a sixth feature, that of simplicity. We consider simplicity an important facet of a solo development environment, since peer review is non-existent in this environment. We use these six features: flexibility, swiftness, leanness, responsiveness, learning and simplicity to compute the agility degrees of each of the SSDM's development core practices and security practices. We compute these as shown in (1).

$$\delta = \frac{1}{n}\sum_{t=1}^{n} x_t; \quad x \in (0, 1) \tag{1}$$

where δ is the agility degree, and x, the value of the agile feature which can hold two values, 1 for existence, 0 for non-existence. To illustrate the use of (1) in computing the degrees of agility, consider the security practice "Security awareness training".

Assessing the existence of the six features in the practice, we assign a 1 to it for flexibility, as we deem it a flexible process, and another 1 for learning, as it promotes learning, we assign a 1 for its being a responsive practice. This is a simple process, so we assign another 1 for this feature. The process however is not fast, so we assign a 0 for speed, and another 0 for it not being lean since it involves some documentation. The score for this practice is 4/6, giving an agility degree of 0.67. The other degrees of the quality and security practices are computed in a similar manner. It should be noted that in inferring the agility values of the agile practices in the SSDM, we also refer to the work of [26] where the agility values of Scrum and XP are computed. This research is similar to these authors' in that most of the SSDM quality practices are drawn from XP and Scrum.

C. CREATING A COMPATIBILITY MATRIX

Having derived the agility degrees of the two sets of practices, we create a compatibility matrix to ease the integration process. The SSDM core-quality practices and security practices can only be integrated if they are compatible [15], [16]. Two practices are compatible if the developer can execute the two simultaneously with minimal loss of productivity. The compatibility between two practices is mainly inferred from the literature and from the close analysis of each practice and what it takes to execute the processes simultaneously, particularly for an individual. Table 3 shows the compatibility matrix with the practices and their associated degrees of agility. A 'C' in a cell in the table shows the compatibility between the two practices whereas 'NC' shows incompatibility between the practices.

D. INTEGRATING QUALITY AND SECURITY PRACTICES

Once the agility degrees of the two sets of practices are computed, and the compatibility matrix is in place, practices can now be integrated. We use Keramati and Mirrian-Hosseinabadi's adapted algorithm which we summarize in the steps below:

1. Choose a security practice with the highest degree of agility.
2. Choose an agile practice with the least agility degree for integration, if none exists, delete the security practice from the list and stop, that is, go to step 6.
3. Integrate the two practices to form a new secure agile practice with agility degree computed as min (a, b) where a is the agility degree of the agile practice and b, the agility degree of the security practice.
4. Check if agility degree of secure agile practice $>= 0.5$. This is in line with the recommendation by Qumer and Henderson-Sellers [27] to consider any practice or methodology with an agility degree $>= 0.5$, as agile. In this case, the research therefore adopts the agility values of 0 to 1 as suggested by these authors. The setting of a threshold of the agility degree to 0.5 is our adaptation of this algorithm.
5. Remove the security practice from the security practices list.
6. Stop or go back to 1 if security practices still exist

To illustrate the use of this algorithm during the integration process, consider an example practice in the list of security practices, that of source code security reviews which had its agility degree computed to be 0.83, and is the highest. this practice is compatible with: coding & code review of with dummy; code refactoring; code integration testing; and task automation. The compatible quality practice with the least agility degree of 0.67 is, coding & code review with dummy. Combining these two gives a minimum agility degree of min (0.67, 083) = 0.67. As this is greater than 0.5, these two can be combined resulting in the practice: Coding & code review, and security code review with the help of a dummy partner. This is a practice in the Development with review phase. The rest of the security practices incorporated into the Secure-SSDM were integrated this way. In Section IV we overview the resulting methodology stages.

IV. THE SECURE-SSDM METHODOLOGY

The Secure-SSDM is an agile methodology designed through the integration of quality practices and security practices drawn from the literature. This gives the methodology its intrinsic quality characteristics. The methodology is organized into six stages: Stage I - Management Buy-in and Standards adoption; Stage II - Functional and Security Requirements Elicitation; Stage III - Release and Sprint Planning; Stage IV - Development with Review; Stage V - Sprint Review and Close, and Stage VI - Evaluation. The stages are summarized as shown in Figure 2. We summarize the practices undertaken in each of these stages in the following sub-sections.

TABLE 3. Compatibility matrix with agility degrees of practices.

Quality practices / Security practices	Security awareness training (0.67)	Security analysis of user roles (0.5)	Misuse case detailing (0.67)	Application of security design principles (0.5)	Security test design (0.67)	Security coding standard adherence (0.67)	Source code security reviews (0.83)	Security testing (0.67)	Security disclosure management (0.67)	Review of security repository (0.83)
User identification (0.83)	C	C	NC	NC	NC	NC	NC	NC	NC	C
User education (0.67)	C	NC	NC	NC	NC	NC	NC	NC	C	NC
Standards Adoption (0.67)	C	NC	NC	NC	NC	C	NC	NC	NC	C
High-level user requirements (0.83)	C	C	C	NC	NC	NC	NC	NC	NC	NC
Prioritization of product backlog (0.83)	C	C	C	NC	NC	NC	NC	NC	NC	NC
Prototype development (0.67)	NC	NC	NC	C	NC	C	NC	C	NC	NC
Creation of user stories (0.83)	C	NC	C	NC	NC	NC	NC	NC	C	NC
Prioritization of Sprint tasks (0.83)	NC	NC	C	NC	NC	NC	NC	NC	NC	NC
Design of acceptance tests (0.67)	NC	NC	C	C	C	NC	NC	NC	NC	NC
Coding prioritized tasks (0.83)	NC	NC	NC	NC	NC	C	C	NC	NC	NC
Version/ change control tracking (0.67)	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
Code refactoring (0.83)	NC	NC	NC	NC	NC	C	C	C	NC	NC
Code review with dummy (0.67)	NC	NC	NC	NC	NC	C	C	NC	NC	NC
Sprint code and quality review (0.67)	NC	NC	NC	NC	NC	NC	C	C	C	C
Project progress review (0.67)	NC	NC	NC	NC	NC	NC	NC	NC	C	NC
Code integration & testing (0.83)	NC	NC	NC	NC	NC	C	C	C	NC	NC
Next Sprint planning (0.83)	NC	NC	C	NC	NC	NC	NC	NC	NC	NC
Deliverables evaluation (0.67)	NC	NC	NC	NC	NC	NC	NC	C	C	C
System acceptance testing (0.83)	NC	NC	NC	NC	NC	C	NC	C	C	C
Task automation (1)	NC	C	NC	C	C	NC	C	C	NC	NC

Key: C – Compatible; NC- Not compatible

STAGE I. MANAGEMENT BUY-IN AND STANDARDS ADOPTION

A developer adopting the Secure-SSDM starts by identifying and educating users on how the project will be undertaken using the methodology. User education in this case applies in cases where a project has identified end users, particularly in an organizational setting. In some cases, a software product is developed for a generic set of users. In this case, the developer

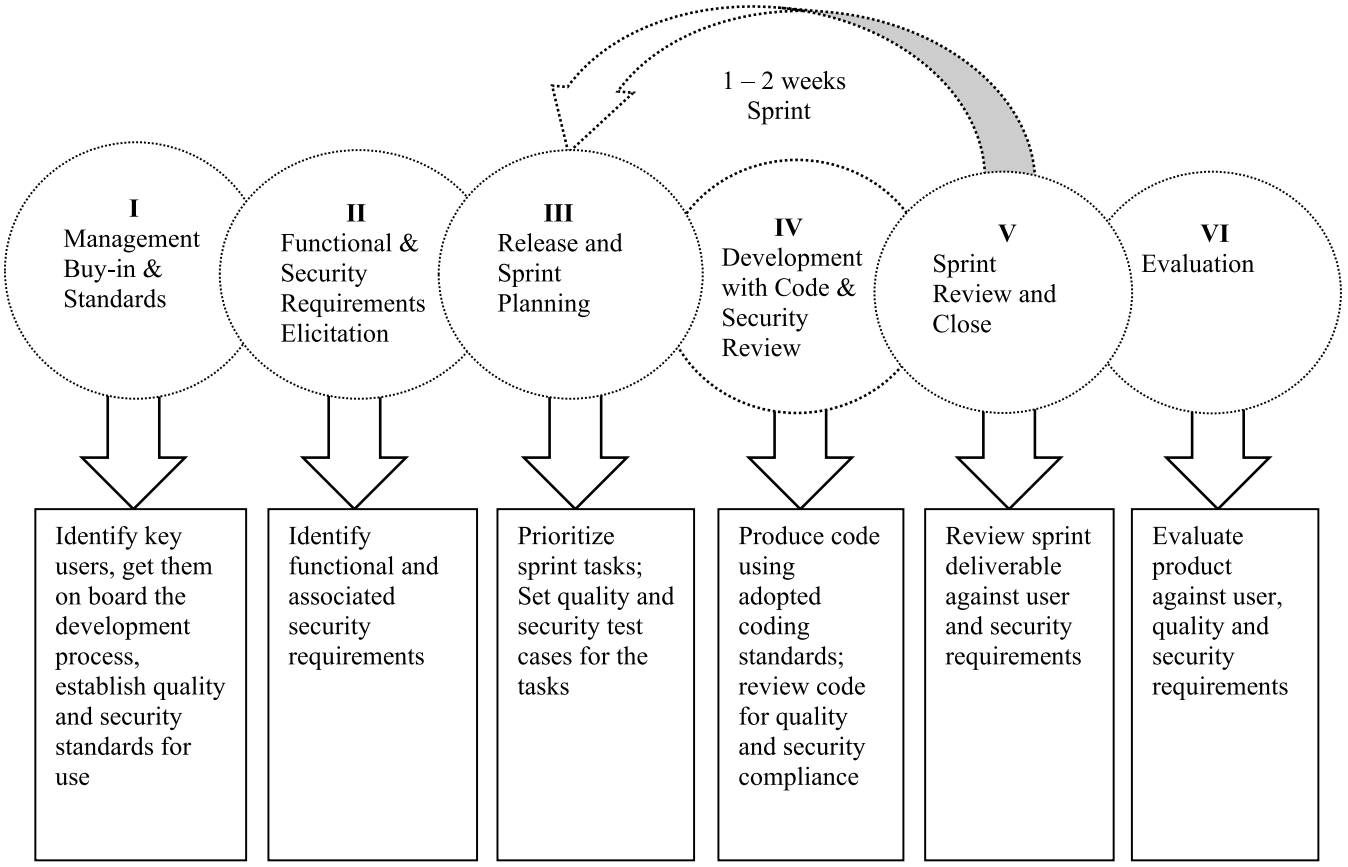

FIGURE 2. Secure-SSDM stages and main activities in each stage.

uses their creativity to formulate requirements for the software. A representative set of clients may be identified to play the role of users if possible. Identifying users at the onset of a software project is a well-stablished concept of user involvement in software development. In addition to educating users on how the development process will proceed, the developer conscientizes users on issues of security. During this period, the developer also adopts appropriate standards (both developmental and security) determined by the type of software under development for use in the project. Developers are encouraged to refer to quality and security standards relevant for their software product to benchmark the quality of the product under development.

It is also important at this point to carry out security analysis on identified user roles. At this point the analysis identifies security threats based on high-level user requirements. The use of an automated dashboard is recommended here to capture and keep track of user requirements, and their associated threats. Bernabé, Navia, and García-Peñalvo [10] recommend the use of tools like Trello or Taiga for single development environments. Trello as a tool enables the developer to organize and manage their work so that they can easily visualize tasks in progress, pending, done, to be reviewed and so on [10]. As a web-based tool it provides for portability enabling the developer to access their dashboard from anywhere.

STAGE II. FUNCTIONAL AND SECURITY REQUIREMENTS ELICITATION

During requirements elicitation, the developer works with users to collect both functional and security requirements. This can be done through collecting user stories describing the users' expected interactions with the system. User stories have to be defined to be small enough so that the developer can work on them independently. These are what users perceive as value addition activities to be obtained from the system. These expected interactions are used to create use case diagrams. We recommend the use of UML diagrams in modelling use cases. At the same time the developer also collects unexpected interactions that may be carried out by unauthorized persons on use cases. These are better identified with the help of the user as they understand the operational environment better.

Unexpected interactions are modelled as misuse cases [28]. The developer should make all effort to get the user to imagine all circumstances that could disturb the smooth flow of their processes. One way to help the user to identify misuse cases is to ask the user to imagine what an intruder would do if they accessed the system. A composite model of use cases and misuse cases is created to show users' interactions with the system. The use cases and their associated misuse cases are used to create a prioritized product backlog. A prioritized product backlog lists the user's expected deliverables

in each development iteration. Capturing misuse cases at such an early stage helps the developer to prioritize security, as opposed to having to think about security at the implementation and testing stages. For complex systems, the developer may build prototypes to facilitate communication between the users. This also helps the developer to understand the system. Once the prioritized backlog is in place, and agreed on by the developer and client, sprint planning may begin.

STAGE III. RELEASE AND SPRINT PLANNING

Release and sprint planning involve identifying the task at the top of the prioritized product backlog, and its component tasks. Since the developer is working on their own, drawing items from an already prioritized product backlog, they can use the logic of the system to order tasks in a sprint. The ordered tasks in a sprint should have time estimates attached to them. The developer should set short iterations of 1 – 2 weeks, to enhance process visibility. Short iterations also motivate the developer at the same time promoting productivity. Appropriate designs for the sprint deliverable should be made, together with security designs and acceptance tests. Sequence diagrams can be used to model the interactions of the user with the system. Another alternative is to use class diagrams. It is advised that the developer keeps models as minimal as possible to enhance productivity. Key points of intruder interceptions should be indicated so as to enable the developer to design means to counter these.

STAGE IV. DEVELOPMENT AND REVIEW

Development with review builds the software product. The developer produces code for the tasks taking care to adhere to coding and security standards adopted at the onset of the project. We recommend the use of version/change control tools to keep track of any code changes. To minimize code defects, we recommend that the developer performs code and source level security reviews with the help of a dummy partner. Identifying errors in the code helps to ensure that only safe and correct code is integrated into the live environment. A dummy partner is any object that the developer may obtain to explain code to. This may also mean practicing self-dialogue. Here the developer runs through all sections of code and explains their functionality to the dummy. This helps to identify errors in code. All identified errors should be fixed before the code is integrated into the baseline.

In producing code, secure coding practices such as avoidance of unsafe functions discussed in [29] should be adopted. These include the avoidance of the use of the strcpy, strncpy, and strcat families of functions from C and C++ programming languages, among others. Instead developers should explore safer versions of functions in their development environments and adopt these. The code at this stage should be secure, with minimal, if not free of coding errors. This is ready for installation at the user's site in the next stage.

STAGE V. SPRINT REVIEW AND CLOSE

Review and sprint close marks the end of a sprint initially set to deliver some functionality (or some component) at the user's site. Here the developer should review the time taken to complete the sprint, against the initial time set for the purpose. Any differences should be noted and used to adjust estimates for the next sprint or future projects. The developer also reviews code quality against set standards. Finished task (s) are moved to completed tasks and incomplete tasks are carried over to the next iteration. The developer also performs code integration, integration testing and security testing. The use of a version control system is highly recommended and so is the use of automation tools discussed in [30]. These include tools used to: quickly access all recently modified code files; correct the most recent commit; delete the most recent commit; and divide a commit in the event the developer detects or suspects some conflict within code components. Such tools help the developer to quickly access the most recent work and perform corrections without taking time to browse all files. Security tests should be performed on all code before integration. At this point the developer also reviews project progress, after which they plan for next sprint or close the project, if no items remain in the backlog.

STAGE VI. EVALUATION

The last stage, Evaluation, concludes the project. This involves evaluation of product deliverables against the user's expectation as well as standards adopted at project onset. This process may be conducted through carrying out a system acceptance test. The system is tested for both user requirements and security requirements fulfilment. At this point the developer should update their development knowledge base as well as security repository. The security repository captures practices to build security into the software. This knowledge management promotes learning in this environment. The developer is also recommended to identify processes or tasks for automation. These are recurring activities within the project life cycle. Automating development processes improves developer productivity [10], [11]. Although most of the Secure-SSDM stages are illustrated as sequential, it should be noted that this is an iterative process, and some tasks may be executed in parallel depending on the size and complexity of the tasks at hand. Table 4 shows the practices in each stage with associated tools recommended for executing the practice. This is meant to address the need of novice developers as was observed with student participants in a case study designed to evaluate the utility of the methodology. Since quality and security are usually not a key concern with most solo developers, we provide here recommended tools for use in executing the practices. We have placed the tools adjacent to the practices to assist the developers.

V. EVALUATION OF THE SECURE-SSDM

The success of DSR is premised on the successful demonstration of the utility of the resulting artefact in addressing the

TABLE 4. Secure-ssdm and recommended tools, techniques and standards.

Stage	`	Tools/Techniques/Standards
I Management Buy-in and Standards Adoption	▪ Education of users on the methodology & institution of security awareness programs ▪ Adoption of development and relevant security standards ▪ Identification of users & security analysis of user roles ▪ Establishment of high-level user requirements	▪ Workshops ▪ Software Quality standards ▪ Requirements checklist (automated, e.g. (Trello or Taiga/ manual dashboard)
II. Requirements Elicitation	▪ Creation of use cases and misuse cases ▪ Creation of a prioritized product backlog ▪ Creation of a WBS (up to task/subtasks) ▪ Categorization of subtasks ▪ Development of prototypes	▪ UML diagrams ▪ Prioritized product backlog ▪ Project management tools (e.g. Project Libre)
III. Release and Sprint Planning	▪ Prioritization of sprint tasks ▪ Designing of quality and security acceptance tests ▪ Attachment of size and time estimates to tasks ▪ Setting of the iteration duration (1 – 2 weeks)	▪ Prioritized sprint backlog ▪ User acceptance tests (short statements showing what the system should do and not do to be acceptable)
IV. Development with Review	▪ Development of code for the tasks taking care to adhere to coding and security standards ▪ Use of version or change control tools ▪ Performing unit tests ▪ Performing of code and security code reviews with the help of a dummy partner ▪ Fixing identified errors ▪ Reviewing time estimates using actual times	▪ Code coverage testing tools (e.g. Jacoco) ▪ Version control system (e.g. Git, Trello) ▪ Dummy partner (explain code to a dummy, self-dialoguing)
V. Sprint Review and Close	▪ Reviewing of sprint time & code quality ▪ Movement of finished task (s) to completed tasks ▪ Carrying over undone tasks to next iteration ▪ Reviewing project progress ▪ Planning for next Sprint (or close project) ▪ Performing of code integration, testing and security testing	▪ Self-dialoguing ▪ Dashboard (e.g. Trello) ▪ Continuous integration ▪ White box security testing
VI. Evaluation	▪ Evaluation of product deliverables & security repository update ▪ Conducting of system acceptance test ▪ Identification of processes/ tasks for automation (repeating tasks)	▪ Task/code automation tools

problem for which the artefact is designed [24], [31], [32]. The Secure-SSDM is designed to improve the quality and in particular security of software products designed by individual developers. This is done through the addition of the security practices to those quality practices that were shown to be existent in the current SSDMs. These were functional suitability, maintainability, usability, reliability, and satisfaction. The resultant Secure-SSDM is therefore expected to promote functional suitability, maintainability, usability, reliability, satisfaction and security in the developed software products.

To that effect the Secure-SSDM has gone through a number of iterations to evaluate its utility. During the first iteration, the SSDM quality framework and its quality practices was presented to participants at a Computing post graduate academic seminar for analysis and feedback. The quality practices and how they build quality into the framework were explained to the participants. The participants confirmed the quality practices with few suggestions for improvement. The highlights for improvement were used to refine the SSDM quality framework. The refined SSDM quality framework was also subjected to blind peer review through a conference

paper. Reviewers' comments were used to further refine the framework.

This refined quality framework is discussed in [13], and was used as a basis for the design of the Secure-SSDM discussed in this paper. Since the Secure-SSDM is built on consensus practices, its quality can also be inferred [24]. To evaluate the utility of the Secure-SSDM in developing quality and secure software, an academic case study was used. A case study is cited as one of the means that can be used to demonstrate the utility of an artefact designed through DSR. In section II we heighted a number of authors that have used the case study in evaluating the usability of a methodology. A case study was deemed appropriate in this paper as it allows us to evaluate the methodology in a socio-technical setting.

Thirty-nine participants (undergraduate students studying towards a four-year degree in Computer Science) participated in the study. This lot of students were doing their second year of a four-year honors degree. This group of participants were chosen due to their accessibility to the first author. The author taught the group of students for a semester long course in Societal computing. This was the third time that the author took the course. The course lasted for three months beginning January 2019 up to March 2019. In this course students are normally expected to identify a problem in the community, to which they can provide a computerized solution. Using this requirement of the course, we deemed this group appropriate for the case study. Since the author had access to the participants, it was easy to monitor progress as participants worked on their individual progress.

Prior to the case study, clearance was sought with the university gate keeper, as well as an ethical clearance to conduct the study. At the beginning of the semester, the methodology was explained to the class. Invitation to participate in the case study was extended to the class. Those who volunteered to participate were asked to use the Secure-SSDM to design and implement software products of their own choice as expected from the course. Software products were designed in the areas of Education, Health, Government, Environment and Business. The areas of application were allocated to the students through a raffle, where each participant was made to blindly pick a piece of paper with an application area on it. These application areas were chosen based on the participants' course outline. We did not change the areas, as we deemed these varied enough to test the various capabilities of the methodology.

The methodology stages were explained to the participants, together with the main quality and security practices. Participants were also given copies of the methodology for constant reference. They were then asked to follow the methodology practices in designing their software products. Feedback on the utility of the methodology was collected through a focus group discussion and document analysis. At the end of the semester a focus group discussion was held to collect participants feedback on the usability of the methodology to build quality and secure products. The first author moderated the focus group discussion, while a teaching assistant attached to the class captured responses to the questions posed by the moderator. Questions posed were divided into three sections; General comments on the usability of the methodology; Comments on the practices in each phase; and Suggestions for improvement. A pre-formulated template to capture responses per question was used for the purpose. Responses were captured against questions posed. At the end of the discussion the first author went through the captured data corroborating what they had captured with what the assistant had captured.

Associated project documentation for the software products was also reviewed for models recommended at each and every stage. This was to verify methodology adherence by the students. Besides checking methodology adherence from the participants in their documentation, we also extracted strengths and weaknesses of the methodology as perceived by the participants. Data from the focus group discussion and documentation was extracted into a spreadsheet. This enabled sorting and color coding of the themes emanating from the data.

During the focus group discussions participants acknowledged the importance of analyzing the security aspect of software products in collaboration with users. As one participant put it:

"I had never thought I could model a system from an intruder's perspective indicating what these could do to hinder the smooth use of the system. After realizing the actions, the intruder could perform on the system, I was forced to think of how to counter them, instead of just using the log-on password as a security measure."

Another participant marveled at the idea of the use of a dummy as a peer review partner. Putting it in their own words:

"At first I thought this was crazy, but after getting down to performing the practice, I noted I was able to discover flaws in my system."

The data that we had extracted to Microsoft Excel was analyzed for themes and patterns. We organized these into strengths, weaknesses and suggestions for improvement. These were drawn from our questions and the data.

Table 5 summarizes the participants' comments on the utility of the Secure-SSDM, captured both from the focus group discussion as well as the methodology section of their project documentation. As shown in the table, while participants indicated more strengths of the methodology, they also pointed out some weaknesses. The weaknesses were mainly associated with the multiple skills that the developer has to possess to successfully use the methodology. As a recommendation to deal with the lack of secure development skills participants suggested sub-contracting some practices such as security testing. This however might increase developmental costs, and subsequently erode the developer' profits. For critical systems, developers may have to consider the suggestion.

Some participants also noted that executing the security practices took some considerable time, particularly for those practices that were new to them. This is acceptable from these

TABLE 5. Themes emanating from student participants' comments on the secure-SSDM.

Strengths	Weaknesses	Suggestions for Improvement
Flexibility	Developer is prone to bias	Quality and security standards should be kept to developers and not shared with users
Frequent customer involvement	Security practices are difficult and time consuming to implement	Suggest tools that can be used with practices to ease the development process
Promotion of security is ideal for online applications.	Difficult to engage management in some cases	Outsource security testing for critical systems
High transparency of the product under development	Minimal documentation may render maintenance difficult	
Better customer satisfaction	Creating intermediate artefacts slows down development process	
Ease communication with users.		
Code review with dummy enables early risk identification		
Combining use cases and misuse cases gives the developer an overall picture of the software		

participants as they had limited knowledge of secure coding practices.

On reviewing the associated models of the software products that participants built, we noted most participants had produced the models that were anticipated for their types of products. Most participants had managed to model misuse cases associated with all their use cases. Based on the use cases and misuse cases, participants managed to come up with design models showing the flow of the system, together with counter measures to deal with identified threats. A few participants, however after coming up with their misuse cases, failed to come up with appropriate designs to counter identified intruder actions. Such practice is common among students to fail to follow adopted methodologies as they pace towards the finishing line [33]. We have since improved on the methodology by adding tools required to support the desired quality practices, particularly security practices.

A. THREATS TO VALIDITY

The case study setting for our evaluation is an academic setting made up of students with varied backgrounds. Some students enroll for the Computer Science course after doing some courses in software development. Such students might have used other practices to develop their software, thereby not giving a clear picture of the utility of the Secure-SSDM. To deal with this threat the first author consistently checked development progress of each participant and their adherence to the practices suggested. We had progress reports conducted fortnightly to check the participants' progress, so that each participant could indicate at what stage of the software development process they were in. We also reviewed their project documentation for the expected deliverables of each stage.

The other threat to validity is the fact that the academic setting is quite different from the industry setting, making it difficult to generalize the results of the case study. The student participants have limited time (in this case study 12 weeks, which also have to accommodate other courses), and limited resources, as in some cases students fail to find real clients and end up hypothesizing most of the requirements. To deal with this threat we emphasized that the participants choose projects that could be handled within their normal class times of four hours per week. This also made it easy for us to track their progress. We are also currently evaluating the Secure-SSDM through a multiple industry case study.

VI. CONCLUSION

We have proposed a Secure-SSDM for individual developers that can be used to build quality, and in particular security into software products. The introduction of security promoting practices to the existing quality practices is our main contribution to the solo software development environment. From a Design science perspective, our contribution to theory is the addition of security practices into the solo software development knowledge base. Using an adapted version of Keramati and Mirian-Hosseinabadi's algorithm we have concentrated on incorporating those lightweight practices that are executable by an individual with minimum effort. Our contribution to practice, is the proposed Secure-SSDM, a methodology that can be used by an individual in developing small to medium-sized software products. We have success fully demonstrated the utility of the Secure-SSDM in building high-quality and secure software applications in the areas of Education, Health, Government and Business. This in turn should help improve the quality of mobile and web-based applications which are popular with solo developers in these application areas.

As future work we recommend the addition of other quality practices to our methodology to address the characteristics of

performance efficiency, compatibility, and portability. These are the quality properties in addition to security that were identified not to be supported by the existing SSDMs after we compared the derived quality framework in [13] with the ISO/IEE 25010 [14] quality model high-level characteristics.

REFERENCES

[1] D. S. Cruzes, M. Felderer, T. D. Oyetoyan, M. Gander, and I. Pekaric, "How is security testing done in agile teams? A cross-case analysis of four software teams," in *Agile Processes in Software Engineering and Extreme Programming* (Lecture Notes in Business Information Processing), vol. 283. Cham, Switzerland: Springer, 2017, pp. 201–216.

[2] A. Firdaus, I. Ghani, and S. R. Jeong, "Secure feature driven development (SFDD) model for secure software development," *Procedia Social Behav. Sci.*, vol. 129, pp. 546–553, May 2014.

[3] I. Ghani, Z. Azham, and S. R. Jeong, "Integrating software security into agile-Scrum method," *KSII Trans. Internet Inf. Syst.*, vol. 8, no. 2, pp. 646–663, 2014.

[4] B. Musa and S. Norita, "Systematic review of Web application security," *Artif. Intell. Rev.*, pp. 259–276, 2015.

[5] 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, Jun. 2018.

[6] H. Homaei and H. R. Shahriari, "Athena: A framework to automatically generate security test oracle via extracting policies from source code and intended software behaviour," *Inf. Softw. Technol.*, vol. 107, pp. 112–124, Mar. 2019.

[7] C. Wijayarathna and N. A. G. Arachchilage, "Why Johnny can't develop a secure application? A usability analysis of java secure socket extension API," *Comput. Secur.*, vol. 80, pp. 54–73, Jan. 2019.

[8] J. Zhu, J. Xie, H. R. Lipford, and B. Chu, "Supporting secure programming in Web applications through interactive static analysis," *J. Adv. Res.*, vol. 5, no. 4, pp. 449–462, Jul. 2014.

[9] R. Agarwal and D. Umphress, "Extreme programming for a single person team," in *Proc. 46th Annu. Southeast Regional Conf.*, 2008, pp. 82–87.

[10] R. B. Bernabé, Á. Navia, and J. García-Peñalvo, "Faat: Freelance as a team," in *Proc. 3rd Int. Conf. Technol. Ecosyst. Enhancing Multiculturality (TEEM)*, 2015, pp. 687–694.

[11] Y. Dzhurov, I. Krasteva, and S. Ilieva, "Personal extreme programming—An agile process for autonomous developers," in *Proc. Int. Conf. Softw., Services Semantic Technol. (S3T)*, 2009, pp. 252–259.

[12] L. Ramingwong, S. Ramingwong, and P. Kusalaporn, "Solo scrum in bureaucratic organization: A case study from thailand," in *IT Convergence and Security*, K. J. Kim, H. Kim, and N. Baek, Eds. Singapore: Springer-Verlag, 2017, pp. 341–348.

[13] S. Moyo and E. Mnkandla, "A metasynthesis of solo software development methodologies," in *Proc. Int. Multidisciplinary Inf. Technol. Eng. Conf. (IMITEC)*, 2019, pp. 353–359.

[14] *Systems and Software Engineering-System and Software Product Quality Requirements and Evaluation(SQauRE)-System and Software Quality Models*, document ISO/IEC FCD 25010, 2010.

[15] H. Keramati and S.-H. Mirian-Hosseinabadi, "Integrating software development security activities with agile methodologies," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl.*, Mar. 2008, pp. 749–754.

[16] Sonia and A. Singhal, "Integration analysis of security activities from the perspective of agility," in *Proc. Agile India*, Feb. 2012, pp. 40–47.

[17] Sonia, A. Singhal, and H. Banati, "FISA-XP: An agile-based integration of security activities with extreme programming," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 3, pp. 1–14, Jun. 2014.

[18] J. S. González-Sanabria, J. A. Morente-Molinera, and A. Castro-Romero, "DeSoftIn: Methodological proposal for individual software development," *Rev. Fac. Ing.*, vol. 26, no. 45, pp. 23–32, May 2017.

[19] K. Beznosov and P. Kruchten, "Towards Agile Security Assurance," in *Proc. New Secur. Paradigms Workshop*, 2004, pp. 47–54.

[20] L. B. Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 6, pp. 497–509, Nov. 2014.

[21] B. Hamid and D. Weber, "Engineering secure systems: Models, patterns and empirical validation," *Comput. Secur.*, vol. 77, pp. 315–348, Aug. 2018.

[22] D. Quiñones, C. Rusu, and V. Rusu, "A methodology to develop usability/user experience heuristics," *Comput. Standards Int.*, vol. 59, pp. 109–129, Aug. 2018.

[23] K. Rindell, S. Hyrynsalmi, and V. Leppänen, "Busting a myth: Review of agile security engineering methods," in *Proc. Availability, Rel. Secur. (ARES)*, 2017, pp. 1–10.

[24] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *J. Manage. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, Dec. 2007.

[25] A. Qumer and B. Henderson-sellers, "Crystallization of agility: Back to basics," in *Proc. Int. Conf. Softw. Technol.*, 2006, pp. 121–126.

[26] A. Qumer and B. Henderson-Sellers, "Measuring agility and adoptability of agile methods?: A 4-dimensional analytical tool," in *Proc. IADIS Int. Conf. Appl. Comput.*, Jan. 2006, pp. 503–507.

[27] A. Qumer and B. Henderson-Sellers, "An evaluation of the degree of agility in six agile methods and its applicability for method engineering," *Inf. Softw. Technol.*, vol. 50, no. 4, pp. 280–295, 2008.

[28] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Eng.*, vol. 10, no. 1, pp. 34–44, 2005.

[29] M. Belk, "Fundamental practices for secure software development," in *Proc. Softw. Assurance Forum Excellence Code (SAFECode)*, 2011, pp. 1–51.

[30] V. Driessen. (Nov. 8, 2018). *Git Power Tools For Daily Use*. Accessed: Nov. 16, 2018. [Online]. Available: https://nvie.com/posts/git-power-tools/

[31] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Berlin, The Netherlands: Springer-Verlag, 2014.

[32] P. Kotzé, A. Van Der Merwe, and A. Gerber, "Design science research as research approach in doctoral studies," in *Proc. 21st Amer. Conf. Inf. Syst.*, 2015, pp. 1–14.

[33] D. Strode and J. Clark, "Methodology in software development capstone projects," in *Proc. 20th Annu. Conf. (NACCQ)*, Nelson, New Zealand, 2007, pp. 243–251.

SIBONILE MOYO received the B.Sc. degree from the University of Zimbabwe, in 1992, and the M.Sc. degree in computer science from the National University of Science and Technology, Bulawayo, Zimbabwe, in 2005. She is currently pursuing the Ph.D. degree in computer science with the Department of Computer Science, School of Computing, University of South Africa.

She is currently a Lecturer with the Department of Computer Science, National University of Science and Technology (NUST), Zimbabwe. She has taught undergraduate courses in software engineering, software development and database systems at NUST for the past 15 years. Her research interests are in software engineering, software development methodologies, and developing quality software.

ERNEST MNKANDLA received the degree (Hons.) in electrical engineering, in 1992, the M.Sc. degree in computer science in 1997, the Ph.D. degree in electrical engineering, in 2008, and the master's degree in open and distance learning, in 2016.

He is currently a Professor with the Department of Computer Science, School of Computing, University of South Africa. He has taught in engineering, computer science, information technology, and information systems at various universities in and outside South Africa for more than two decades and has supervised many M.Sc. and Ph.D. students. He is a rated Researcher in South Africa and has passion in the development of quality software and believes in the betterment of humanity through the provision of quality software technologies and seamless synergy between humans and machines. He hopes for a future where there is a balance between new technology innovations and ethics. He therefore researches and publishes extensively in software engineering and artificial intelligence. He has provided consultancy to industry in software development and information technology.