مجلة جامعة بني وليد للعلوم الإنسانية والتطبيقية
Bani Waleed University Journal of Humanities and Applied Sciences
تصدر عن– جامعة بني وليد – ليبيا
Website:https://jhas-bwu.com/index.php/bwjhas/index

ISSN3005-3900 الصفحات (213- 224) المجلد العاشر ــ العدد الرابع ــ 2025

Security Metrics for Assessing Security Risks of Software in Agile Development Methods

Ala A. Abdulrazeg [1]*, Salim A. Adrees [2], Faitouri. A. Aboaoja [3]
[1.2] Computer Engineering Department, Faculty of Engineering, Omar Al-Mukhtar University, El-Beida, Libya.
[3] Computer Science Department, Faculty of Science, University of Derna, Derna, Libya.
ala.ali@omu.edu.ly

مقاييس الأمان لتقييم مخاطر البرمجيات في منهجيات التطوير الرشيقة

علاء علي عبدالرازق[1]*، سالم علي عبدالرازق[2] ، فيتوري عوض الفيتوري[3]
[2.1] قسم هندسة الحاسوب، كلية الهندسة، جامعة عمر المختار، البيضاء، ليبيا.
[3] قسم الحاسوب، كلية العلوم، جامعة درنة، درنة، ليبيا.

تاريخ الاستلام: 06-09-2025 تاريخ القبول: 13-10-2025 تاريخ النشر: 20-10-2025

الملخص:
أصبحت منهجيات تطوير البرمجيات الرشيقة من الركائز الأساسية في تطوير البرمجيات الحديثة، حيث توفر المرونة والتسليم السريع الذي يركز على احتياجات المستخدمين. ويُعدّ دمج ممارسات الأمن ضمن سير عمل هذه المنهجيات أمر بالغ الأهمية لبناء أنظمة قوية وموثوقة. ومع ذلك، فإن الاعتماد على الممارسات الأمنية وحدها لا يوفّر رؤية واضحة وكاملة لحالة الأمان الفعلية للتطبيق. لسد هذه الفجوة، تحتاج فرق الأجايل إلى مؤشرات أمنية قابلة للقياس وقابلة للتنفيذ تمكنها من التقييم المستمر وتحسين ممارسات دمج الأمن ضمن عملية التطوير، وهو أمر يبقى تحدياً في ظل غياب هذه المؤشرات. تقدم هذه الورقة مجموعة منظمة من المقاييس الأمنية صُممت لتقييم فعالية الأنشطة الأمنية عبر المراحل الرئيسية لسير عمل أجايل: قبل وأثناء وبعد كل دورة تطوير. تركز المقاييس المقترحة على العناصر القابلة للقياس مثل شمولية نماذج التهديد، معايير القبول الأمني، الالتزام بممارسات البرمجة الآمنة، نتائج الاختبارات الأمنية، وإدارة الثغرات بعد انتهاء الدورة. تهدف هذه المقاييس إلى تقييم مدى قدرة الفرق على تقديم برمجيات آمنة مع الحفاظ على المرونة في العمل.

الكلمات الدالة: تطوير البرمجيات الرشيقة, الأنشطة الأمنية, مقاييس الأمن, إدارة الثغرات.

Abstract
Agile methodologies have become key players in modern software development, offering flexibility, rapid and customer-focused delivery. Integrating security practices into Agile workflows is critical for building resilient and trustworthy systems. However, security practices alone provide limited visibility into an application's actual security posture. To address this gap, Agile teams require quantifiable, actionable security metrics that enable continuous assessment and improvement of security integration throughout the development process, a task that remains challenging due to the lack of such measures. This paper proposes a set of security metrics designed to evaluate the effectiveness of security activities across key Agile phases: before, during, and after each iteration. The proposed metrics focus on measurable artefacts such as threat model coverage, security acceptance criteria, secure coding adherence, security testing results, and post-iteration vulnerability management. These measures aim to assess how effectively teams deliver secure software while maintaining agility.

Keywords: Agile Software Development, Security Activities, Security Metrics, Vulnerability Management.

Introduction:

Agile software development has become widely adopted due to its flexibility, iterative progress, and responsiveness to changing requirements. By emphasizing collaborative workflows, rapid delivery, and continuous feedback, agile methodologies enable development teams to produce high-quality software that aligns closely with customer needs and market demands (Sinha & Das, 2021). As software increasingly supports critical business functions, ensuring its security has become a top priority. With most business applications now accessible online, the number of security attacks exploiting software vulnerabilities has significantly risen. These attacks have led to substantial losses of personal and financial data, impacting both individuals and organizations (Thool & Brown, 2024). The growing frequency and diversity of cyber-attacks highlight the critical importance of integrating security practices within Agile workflows to mitigate vulnerabilities at early stages (Rindell et al., 2021; Thool & Brown, 2024). Such integration not only helps prevent costly breaches but also preserves user trust in rapidly evolving software products.

Even when secure software development practices are correctly followed from the very beginning, the development team may still lack a clear understanding of the actual security level achieved. This uncertainty can lead to challenges such as those outlined by Vacca (2012):

- How much more do I need to spend to be "safe" from attack?
- Will the changes made to my software to improve security be effective?
- Are my company's workflows or processes sufficiently secure?

To address the above security questions and assess the security state of a system, it is essential to measure it (Abdulrazeg et al., 2012). Security cannot be improved without measurement. To measure security, security metrics is needed (Wagner & Ford, 2020). Security metrics can be regarded as an integral component of both software development and software assurance, to assess whether the software achieves its intended objectives. Well-defined security metrics can serve as an effective tool for the development team to assess the effectiveness of various artefacts, deliverables and documents that are produced during the development lifecycle (Sherif et al., 2024).

Therefore, to effectively strengthen security in Agile environments, development teams should be equipped with a security metrics to monitor and track the implementation of security practices, as well as to assess whether team members possess the necessary security knowledge and training (Ojuri, 2024).Without clear and objective measures, it becomes difficult to evaluate the security posture or identify gaps that may lead to risks (Sherif et al., 2024).

In this paper, we propose a set of security metrics to evaluate the effectiveness of integrated security practices across the key phases of the Agile workflow: before, during, and after each iteration. The proposed security metrics focus on measurable artefacts such as threat model coverage, security acceptance criteria, secure coding compliance, security testing outcomes, and post-iteration vulnerability management. The metrics supports decision-making, promotes early risk identification, facilitates secure coding and testing practices, and encourages continuous learning from retrospective analyses. By quantitatively assessing these metrics, development teams can gain actionable insights into their security posture and iteratively improve security integration throughout the development life cycle. This work addresses the absence of phase-specific security metrics in Agile workflows, and we believe our contribution represents a step toward enabling teams to deliver more secure software without compromising agility.

The rest of this paper is organized as follows: section 2 gives an overview of Agile methodologies. Section 3 reviews security practices in Agile software development. Section 4 discusses related works in the field of security metrics. In section 5, we present the proposed security metrics. Section 6 describes how to integrate the proposed security metrics into a secure Agile development process, while section 7 concludes the paper and outlines future research.

1. Agile methodologies

Agile methodologies have become an increasingly popular software engineering (SE) process over the last decade and widely adopted by software development teams (Nagele et al., 2022). These methods put emphasis on flexible, iterative, and customer-centric processes. Previous research has suggested that Agile

methods enhance team productivity and product quality, foster communication and promote knowledge sharing (Thool & Brown, 2024). Numerous SE processes apply Agile methodologies, including Scrum, Extreme Programming (XP), Kanban, Feature-Driven Development (FDD), and Dynamic Systems Development Method (DSDM) (Thool & Brown, 2024). While each method has its own principles, life cycle, and roles, all agile methods build the software in iterative process (Al-Saqqa et al., 2020). While Scrum employs fixed-length time-boxed iterations called sprints, other models refer to their development cycles more generally as iterations. Despite differences, these models can all be viewed through the lens of iteration phases, which can be effectively structured into three distinct phases:

- Before-iteration (Planning and Preparation)

This phase focuses on clarifying requirements that are expressed as user stories and features, defining acceptance criteria for each item, prioritizing backlog items based on business value, cost, or risk. Capacity and effort estimation is required for each prioritized item. Setting iteration goals provides clarity on what is expected during the iteration, making it easier for team members to communicate and collaborate effectively.

- During-iteration (Execution and Progress Tracking)

This phase focuses on the execution period where the team develops, tests, and delivers working software based on planned user stories and features. Key activities include coding using best practices such as test-driven development, continuous testing to maintain quality, and daily stand-ups for synchronization and issue resolution. This iterative process ensures continuous delivery of functional software, with progress tracked through visual tools like task boards or burndown charts.

- After-Iteration (Review, Feedback, and Retrospective)

This phase focuses on reviewing completed work with stakeholders, conducting acceptance testing to verify that user stories meet defined criteria, and gathering feedback. The team conducts an iteration retrospective to reflect on successes and challenges, identifying improvements for future iterations.

2. Security Practices in Agile Software Development

To be considered secure software that performs its intended functions correctly, the software must be free from vulnerabilities. Improving security and achieving quality software requires the early integration of security practices into the development lifecycle (Mohaddes et al., 2015; Abdulrazeg & Abdulrrzaiq, 2020). However, there is no standardized set of security practices that can be implemented in software development projects (Vicente et al., 2019). Moreover, the project team usually has limited security expertise and experience, along with a general lack of security awareness (Vicente et al., 2019). Despite these challenges, the positive results of integrating security early in the development process include (1) reduced security issues, (2) quicker defect rectification, (3) highlighting risks in the early phases of the development process, and (4) reduced costs (Mohaddes et al., 2015).

In the literature, relevant studies have integrated security practices into agile methods to develop secure agile software development models (Maier et al., 2017; Rindell et al., 2018; Vicente et al., 2019; Bezerra et al., 2020; Moyon l et al., 2020).

Thool and Brown conducted a comprehensive review of existing literature on Securing Agile software development and identified eight security practices as being particularly valuable in improving the security of Agile software development processes (Thool & Brown, 2024). These security activities are;

- Addressing security in early iterations with requirements and testing before deploying the software.
- Incorporating security expectations in project requirements when describing the responsibilities and behaviour of the software.
- Adding a security specialist such as a Security Master to the development team to focus on addressing security concerns and ensuring system security.
- Increasing story points or weights to prioritize security-related tasks and encourage more secure development and testing.
- Incorporating vulnerability and penetration testing such as Dynamic Application Security Testing (DAST) to automatically detect security flaws in running software.
- Using static analysis tools such as Static Application Security Testing (SAST) to detect security vulnerabilities by automatically scanning the source code.
- Performing risk analysis to identify vulnerabilities.

- Integrating secure coding practices into the deployment pipeline to enable automatic security checks with code changes and address issues before deployment.

Ardo et al. identified security practices through interviews with agile practitioners aimed at securing the Agile development process and reducing cybersecurity breaches (Ardo et al., 2022). These practices include,

- Involve security specialist and penetration tester as integral members to the development team.
- Define evil user stories through threat modelling session and regularly reviewed them after each iteration to better understand threat scenarios and malicious behaviours.
- Conduct brainstorming sessions to assess the feasibility and implementation of proposed security features.
- Generate security test plans derived from the security backlog to ensure software meets defined security features.
- Perform security regression tests to verify that recent code changes do not introduce new vulnerabilities.
- Conduct a secure code review session to identify security vulnerabilities through manual inspection and automated static analysis tools.
- Perform manual and automated penetration testing alongside secure code review sessions to identify exploitable vulnerabilities beyond static analysis.
- Hold a Security Retrospective at sprint's end to review what security practices worked well, uncover gaps or failures, and decide on actionable improvements for upcoming development cycles

The preceding section emphasizes the importance of integrating security practices, such as threat modelling, secure code reviews, penetration testing, and regression tests within Agile development methods to improve security and deliver high-quality, vulnerability-free software. However, implementing these activities alone does not always provide clear insight into how effectively security goals are met or reveal existing gaps. The current focus largely overlooks the systematic use of security metrics to measure and track security effectiveness. Therefore, security metrics play a crucial role in measuring effectiveness, guiding risk-aware decisions, and enabling continuous security improvement throughout the Agile lifecycle.

3. Related Works

Several studies have proposed the use of security metrics to evaluate software security during the development life cycle. Sultan et al., (2008) proposed a set of metrics to monitor the security level of the software throughout the entire development process, providing engineers with early feedback before release. Savola et al. (2012) introduced a risk-driven methodology that guides agile teams in deriving project-specific security metrics by linking high-level goals to questions and detailed measures through agile risk analysis activities. Abdulrazeg et al. (2012) and Kundi and Chitchyan (2014) agreed that measuring security by focusing on the misuse case model allows designers to detect and fix security vulnerabilities early in the development process, preventing them from reaching the final product. OWASP Software Assurance Maturity Model (SAMM) (OWASP, 2020) emphasizes the need to define metrics with insight into the effectiveness and efficiency of the application security program. The model identifies measurable indicators across three categories: effort (e.g., training hours, code review time, number of apps scanned), results (e.g., unresolved security defects, security incidents), and environment (e.g., number of applications, total lines of code). Wagner and Ford (2020) proposed metrics to support Agile Software Development in regulated environments. The authors identified metrics to measure performance of three regulatory attributes: software quality assurance (defect density, code coverage), security (security test pass rate, code scanning, detection rate), and software effectiveness (earned business value, tasks completed, tasks with errors). Caniglia et al. (2025) introduced FOBICS, a quantitative metrics framework designed to measure the efficiency of using DevSecOps, considering both security and business logic perspectives. The framework integrates security-related indicators such as Number of total tests, Performance Security Index, Number of workers with a specific permission, Number of code commits made before the first test is executed, Security Coverage Index and Training Security Index.

The review of existing literature shows that there is no widely accepted model for measuring security within agile methodologies. In particular, little work has focused on developing security metrics that are

explicitly aligned with the different phases of agile methodologies. Therefore, this gap in knowledge is what our study aims to fill.

4. Security Metrics for Agile Software Development

In this work, a security metrics are developed to be applied at the key phases of the Agile workflow: before, during, and after each iteration, aiming to assess and detect software security risks. The objective of the security metrics is to monitor and track the implementation of security practices and to assess whether team members possess the necessary security knowledge and training. The importance of this approach comes from the fact that assessing security risks early in the development life cycle can help the team to implement efficient solutions before software delivery to customers (Thool & Brown, 2024). Below are the proposed security metrics, grouped according to each phase of the agile iteration to ensure continuous security vigilance. The proposed security metrics is developed based on the practices and activities outlined in Thool and Brown (2024) , Ardo et al. (2022), and Ojuri (2024) .

- **Before Iteration (Planning phase)**

Metrics 1.1: *The ratio of the number of team members completing relevant security training to the total number of team members [RTMST].*

This metric quantifies the number of team members who have completed relevant security training, thereby reflecting the team's security readiness and the organization's commitment to security culture. RTMST is crucial for ensuring that development teams possess up-to-date security knowledge, a prerequisite for effectively embedding security into agile methodologies and mitigating potential vulnerabilities. Consider a number of team members as $TTM = (ttm_1, \ldots, ttm_n)$ and the team member with security training as $TMST = (tmst_1, \ldots, tmst_n)$ such that $TMST \subseteq TTM$. The metrics is defined as follows;

$$RTMST = TMST / TTM$$

The value of the metric range from [0 - 1], if RTMST converges to 1, it indicates all team members have completed the relevant training and reflects strong security awareness. The lower RTMST value suggests that many team members lack essential security knowledge. This situation poses potential security risks, as untrained members may unintentionally introduce vulnerabilities or fail to adequately identify and mitigate potential threats. The lower RTMST serves as an indicator for the management to prioritize training initiatives aimed at equipping untrained members with security knowledge. Moreover, it is imperative for management to systematically monitor training progress to ensure and foster a culture of security awareness.

Metrics 1.2: *The ratio of the number of user stories with threat models to the total number of user stories in the backlog [RUSTM].*

This metric measures the extent to which threat modelling is applied to user stories in the backlog during backlog refinement. It reflects early integration of security analysis in agile development to ensure the reduction of security vulnerabilities before coding begins. Consider a set of user stories in the backlog as $TUS = (tus_1, \ldots, tus_n)$ and the user stories with threat models as $USTM = (ustm_1, \ldots, ustm_n)$ such that $USTM \subseteq TUS$. The metrics is defined as follows;

$$RUSTM = USTM / TUS$$

The value of the metric range from [0 - 1], if *RUSTM* converges to 1, it indicates strong early security integration where most user stories have undergone threat modelling. This suggests proactive identification and mitigation of security risks before development. The lower *RUSTM* value implies limited or no threat modelling on backlog items, indicating reactive security posture. This increases risk of undetected threats entering development, potentially causing costly fixes later. Therefore, to overcome the lower RUSTM, introduce threat modelling criteria in definition of ready, involve security experts in backlog refinement, and train teams on threat modelling.

Metrics 1.3: *The ratio of the number of user stories with defined security acceptance criteria to the total number of user stories in the backlog [RUSAC].*

This metric quantifies the number of user stories in the backlog that incorporate clear and testable security-related acceptance criteria that must be met for the user story to be considered complete. This metric reflects how well security is integrated into agile planning artefacts, enabling early risk mitigation. Consider a set of user stories in the backlog as $TUS = (tus_1, \ldots, tus_n)$ and the user stories with defined

security acceptance criteria as $USAC = (usac_1, \ldots, usac_n)$ such that $USAC \subseteq TUS$. The metrics is defined as follows;

$$RUSAC = USAC / TUS$$

The value of the metric range from [0 - 1], if RUSAC converges to 1, it indicates security requirements are well integrated into the backlog, ensuring that security considerations are explicitly part of the development scope. The lower RUSAC value implies limited or no integration of security acceptance criteria, increasing the risk of security gaps and vulnerabilities being introduced during development. Therefore, to overcome the lower RUSAC, immediate focus is needed to embed security acceptance criteria into backlog items and increase this ratio over time to improve security coverage. The security acceptance criteria should be specific, relevant, measurable, and testable to be effective.

Metrics 1.4: *The ratio of the number of security requirements aligned with security acceptance criteria of the user stories to the total number of identified security requirements. [RSRAC].*

The security requirements alignment ratio measures the extent to which identified security requirements are explicitly linked and reflected in the security acceptance criteria of user stories. This metric evaluates how well security requirements translate into actionable, testable acceptance criteria that guide development and testing. Consider a number of security requirements as $TSR = (tsr_1, \ldots, tsr_n)$ and the security requirements aligned with security acceptance criteria as $SRAC = (srac_1, \ldots, srac_n)$ such that $SRAC \subseteq TSR$. The metrics is defined as follows;

$$RSRAC = SRAC / TSR$$

The metric RSRAC ranges from [0 - 1]. A value approaching 1 indicates a strong alignment between security requirements and acceptance criteria, ensuring that security requirements are effectively translated into specific, testable conditions. This alignment facilitates robust implementation, verification, and traceability, thereby reducing the likelihood of security gaps. Conversely, a lower RSRAC value suggests that some or many security requirements lack associated acceptance criteria. Such misalignment increases the risk that critical security requirements may be overlooked or insufficiently tested, potentially resulting in insecure features or vulnerabilities in the final product. To mitigate these risks, it is essential for cross-functional team including security experts, developers, and product owners to collaboratively map each security requirement to one or more user stories with well-defined, measurable security acceptance criteria. Ensuring this proper alignment guarantees that security requirements are not only documented but also validated through security testing, significantly decreasing the probability of defects and security vulnerabilities slipping into production.

Metrics 1.5: *The ratio of the security acceptance criteria that are covered by security test cases to total number of identified security acceptance criteria [RSATC].*

This metric is applied to ensure that security test cases are developed for each security acceptance criterion associated with user stories. By measuring RSATC, development teams can proactively validate that security considerations are not only identified but also operationalized through executable tests, enabling effective verification and decision-making about product acceptability from a security perspective. Regular monitoring of this metric helps development teams identify and close gaps in security testing coverage, thereby enhancing overall security quality. Consider a number of identified security acceptance criteria as $TAC = (tac_1, \ldots, tac_n)$ and the number of security acceptance criteria covered by security test cases as $SATC = (satc_1, \ldots, satc_n)$ such that $SATC \subseteq TAC$. The metrics is defined as follows;

$$RSATC = SATC / TAC$$

The metric RSATC ranges from [0 - 1]. A value approaching 1 indicates nearly all security acceptance criteria are covered by test cases, which reflects high level of security assurance. On the contrary, a lower RSATC value suggests that some or many security acceptance criteria lack associated test cases. This implies that certain security requirements remain unverified, signalling potential gaps in validation and increasing the risk that security flaws may go undetected. The lack of test coverage undermines confidence in the product's security posture. Therefore, it is essential to apply shift-left testing principles by incorporating security test case development early during backlog refinement.

Metrics 1.6: *The ratio of the user stories with high exposure to security risks to total number of user stories [RUSHR].*

This metric measures the number of user stories that involve high exposure to security risks such as handling sensitive data, performing authentication, or accessing critical system resources relative to the total number of user stories in the backlog. Security acceptance criteria and story points assist the team in identifying which user stories carry higher security exposure. This metric helps the team to anticipate challenges and allocate efforts effectively. This supports better decision-making, improves user story quality by encouraging clearer security acceptance criteria for high-risk stories, and contributes to maintaining overall project health through proactive risk management.

Consider the number of user stories as TUS = (tus_1,…, tus_n) and a number of user stories with high exposure to security risks as USHR = ($ushr_1$,…,$ushr_n$) such that $USHR \subseteq TUS$. The metrics is defined as follows;

$$RUSHR = USHR / TUS$$

A higher RUSHR indicates the presence of complex, high-risk user stories, signalling the need for focused attention on thorough definition, rigorous security acceptance criteria, and targeted risk mitigation efforts to ensure quality and reduce ambiguity of the user stories and their security requirements. Conversely, a lower RUSHR reflects a backlog centred on lower-risk stories, indicative of more stable project conditions and potentially higher overall quality and predictability in delivery, thereby facilitating more effective resource management.

- **During Iteration (Development phase)**

Metrics 2.1: *The ratio of secure code commits that meet defined secure coding criteria to total number of code commits during iteration [$RSCSC_i$].*

This metrics measures the ratio of code commits during an iteration that adhere to secure coding practices verified by passing automated static analysis (SAST) validation tools, relative to the total number of commits. RSCSC reflects the effectiveness of secure development practices at the implementation level. It serves as an early indicator of potential security debt being introduced into codebase. Additionally, RSCSC Supports continuous improvement by tracking and analysing secure coding compliance across successive sprints ($rscsc_1$,…,$rscsc_n$).

Consider a number of code commits during an iteration as TCC = (tcc_1,…, tcc_n) and the number of secure code commits introduced during that iteration as SCSC = ($scsc_1$,…,$scsc_n$) such that $SCSC \subseteq TCC$. The RSCSC for a specific iteration *i*, is calculated as:

$$RSCSC_i = SCSC / TCC$$

The ratio of secure code commits (RSCSC) metric ranges from 0 to 1. A value approaching 1 indicates a strong adherence to secure coding practices, significantly reducing the likelihood of introducing vulnerabilities during iteration's development process. Conversely, a lower RSCSC value suggests that many commits do not comply with secure coding standards, increasing the risk of security flaws in the codebase. To mitigate this risk, it is essential to enforce clear secure coding guidelines and policies for developers to follow. Additionally, providing regular training on secure coding practices and common vulnerabilities is crucial to enhance developer skills and raise security awareness.

Metrics 2.2: *The ratio of code changes that have been reviewed with security-focused during iteration, relative to the total number of code changes made [$RCRSF_i$].*

This metric measures the number of code changes in an iteration that have undergone manual peer-review with explicit attention to security considerations before being merged into the main codebase. Peer-reviews are essential because they can identify subtle security issues that static or dynamic analysis tools may overlook. This effectively complements automated testing efforts, thereby reducing overall vulnerability density in the codebase.

Consider a number of code changes during an iteration as TCG = (tcg_1,…,tcg_n) and the number of code changes with security-focused peer review as CRSF = ($crsf_1$,…,$crsf_n$) and such that $CRSF \subseteq TCG$. The RCRSF for a specific iteration *i*, is calculated as:

$$RCRSF_i = CRSF / TCG$$

The ratio of code review coverage (RCRSF) metric ranges from 0 to 1. A value approaching 1 indicates nearly all code changes are reviewed with security in mind during an iteration. Conversely, a lower RCRSF value suggests that many code changes are merged without proper security-oriented peer-review. To address a low RCRSF value, it is essential to establish clear criteria defining what constitute a

thorough security-focused review, such as using standardized checklists and requiring explicit security sign-offs. Security reviews should be integrated as a mandatory step within each iteration, ensuring that no code is merged without explicit security review. Providing targeted training to the team enhances their ability to effectively identify security issues during peer reviews.

Metric 2.3: *The ratio of security-related defects identified during an iteration to the total number of story points completed during that iteration [$RSDSP_i$].*

This metric measures the density of defects relative to the amount of work completed in an iteration *i*, where work is estimated in story points. Since story points represent the relative effort and complexity of user stories, this metric provides a normalized view of software quality by showing how many defects occur per unit of delivered effort. RSDSP helps development teams evaluate their effectiveness in managing complexity, testing, and defect prevention, while encouraging a balance between delivery speed and quality. By tracking defect density over time, teams can identify trends in relation to their workload, prioritize high-risk areas for additional testing or refactoring, and focus on continuous process improvement.

Consider a number of story points completed in an iteration as TSP = (tsp_1,…, tsp_n) and the number of security-related defects detected through manual and automated security testing, and security peer review during that iteration as SDSP = ($sdsp_1$,…,$sdsp_n$). The RSDSP for a specific sprint *i*, is calculated as:

$$RSDSP_i = SDSP / TSP$$

The value of the metric range from [0 - 1], if RSDSP converges to 0, it indicates that few security-related defects are detected per story point, reflecting strong security practices and effective prevention of vulnerabilities early in the development process. Conversely, a high RSDSP signals an increased density of security defects relative to the amount of work delivered, indicating potential weaknesses in design, coding, or testing. To overcome a high RSDSP ratio, teams should integrate security earlier in their workflows. Additionally, on-going secure coding training enables team members to develop the skills necessary to write secure code and avoid common security defects.

Metric 2.4: *The ratio of security-test cases failed to the total number of security test cases executed during an iteration [$RSTCF_i$].*

This metric determines the number of security test cases that detect implementation defects, measured as the number of failed security test cases relative to the total number executed security-test cases (both passed and failed) within an iteration. It serves as an indicator of the security robustness of the software under test, reflecting the extent to which security vulnerabilities are exposed during testing. Consider a number of security test cases executed during an iteration as TCE = (tce_1,…, tce_n) and the number of security test cases failed during that iteration as STCF = ($stcf_1$,…,$stcf_n$), such that $STCF \subseteq TCE$. The $RSTCF_i$ for a specific iteration *i*, is calculated as:

$$RSTCF_i = STCF / TCE$$

The ratio of security test cases that fail ranges from [0 to 1]. A value approaching 0 indicates that most security tests have successfully passed, reflecting strong security practices and less vulnerabilities detected during an iteration. Conversely, a high RSTCF value signals that many security tests failed, highlighting potential weaknesses in the code or security controls that require immediate attention. To address a high RSTCF value, team should analyse the failed test cases to identify root causes and fix high-severity vulnerabilities. Additionally, Improving test coverage, integrating security-focused code reviews, and leveraging automated security testing can help prevent defects and gradually raise the pass ratio.

- **After Iteration (Retrospective phase)**

Metric 3.1: *The ratio of defects that escape detection during an iteration to the total number of defects found during and after an iteration [$RDEDI_i$]*

This metric measures the number of defects that are not detected during an iteration phase but are subsequently detected either during the post-sprint phase or user acceptance testing (UAT), relative to the total defects found during and after an iteration. This metric provides critical insight into the effectiveness of in-iteration testing and quality assurance processes in identifying and fixing issues early thereby helping to assess defect leakage and areas for process improvement.

Consider a number of defects found during and after iteration as TDF = (tdf_1,…, tdf_n) and the defects escaped detection during an iteration as DEDI = ($dedi_1$,…,$dedi_n$), such that $DEDI \subseteq TDF$. The metrics is defined as follows,

$$RDEDI_i = \text{DEDI} / \text{TDF}$$

A low defect escape ratio indicates effective testing and quality assurance process, where most defects identified and resolved during iteration and before release. This suggests that current practices are robust, and continuous improvement in automation and coverage should be maintained. Conversely, a high defect escape ratio reveals that the defects are leaking through to users or production, signalling insufficient security test coverage, inadequate automation, or weak security review practices.

Metric 3.2: *The ratio of security-related improvement actions identified in the previous iteration retrospective that have been implemented to the total security improvement actions documented [RSAIR].*

This metric measures the number of security-related improvement actions identified in the previous iteration retrospective that have been successfully implemented within the current iteration. It serves as a critical indicator of the team's commitment to continuous security improvement by acting upon lessons learned to enhance process and product quality. Consider a number of security-related improvement actions identified in retrospectives as TSI = (tsi_1,…, tsi_n) and the security-related improvement actions implemented as *SASR* = ($sasr_1$,…,$sasr_n$), such that $SASR \subseteq TSI$. The $RSASR_i$ for a specific iteration *i*, is calculated as:

$$RSASR_i = SASR / TSI$$

A higher RSASR demonstrates that the team effectively executes most of the security actions identified during retrospectives. This reflects a culture of continuous learning, prioritization of security within the development workflow, appropriate resource allocation, and strong leadership support. Conversely, a low ratio suggests many identified issues remain unaddressed, indicating weaker commitment. To overcome a low RSASR ratio, team should prioritize security actions in an iteration planning and add the most important actions to the next iteration's backlog so they are treated as actual work items.

5. Integrating Security Metrics into Secure Agile Process

Applying the proposed security metrics within a secure agile method offers a structured approach to illustrating security measurement across the development cycles. To demonstrate the application of the proposed security metrics, the model is integrated with the Secure Scrum process proposed by Maier et al. (2017). Figure 1 shows the alignment of the security metrics model with the three key phases of the Secure Scrum process: before, during, and after sprint. This integration demonstrates how security metrics can be systematically incorporated into agile workflows to support continuous security improvement.

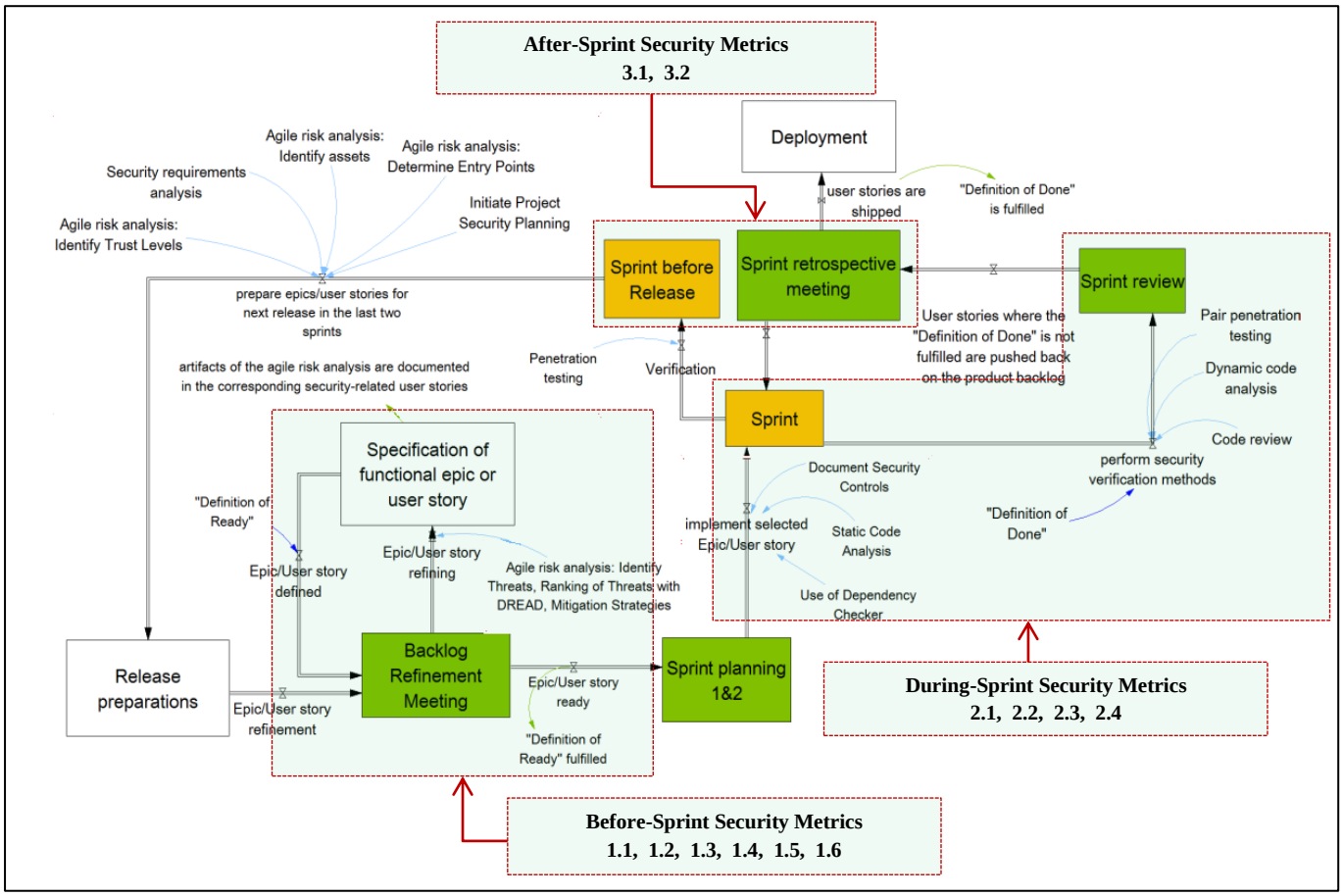

Figure 1: Integrating Security Metrics Model into Secure Scrum process proposed by Maier et al. (2017)

6. Conclusion and Future Work

This paper proposes a set of security metrics designed to evaluate software security risks within Agile development methods, covering key phases: before, during, and after each iteration. The objective of these metrics is to provide actionable insights that enable development teams to continuously assess and improve security practices across the Agile process, thereby strengthening their overall security posture. Quantitative indicators such as threat model coverage ratio, secure code commit compliance, and defect leakage rates measure the effectiveness of security implementation. A high coverage ratio or low defect leakage rate indicates successful adoption of secure practices, supporting the objectives of this study. Additionally, the proposed metrics include indicators assessing the effectiveness of security training, providing valuable insights into organizational attitudes and behaviors toward security awareness, which is a critical factor influencing the overall security posture.

Beyond evaluating individual aspects of security, the proposed metrics support a continuous cycle of measurement and improvement, showing how ongoing evaluation across Agile phases enhances software security. For instance, security metrics that track failed security test cases and security-related defects provide concrete evidence of system vulnerabilities. These findings are analyzed during the retrospective phase to identify actionable improvements, which are then incorporated in subsequent iterations, reinforcing security practices and sustaining the feedback loop.

Future work will involve conducting experiments of applying these metrics to a variety of software projects to collect quantitative data and qualitative feedback. These experiments aim to evaluate and demonstrate the usefulness and effectiveness of the proposed security metrics in supporting secure system development.

7. References

Abdulrazeg, A. A., & Abdulrrzaiq, S. A. (2020). A Hybrid SV-SCRUM Model: Integrating SCRUM in SV-Model. Journal of Pure & Applied Sciences, 19(5), 90-95.

Abdulrazeg, A. A., Norwawi, N. M., & Basir, N. (2012, June). Security metrics to improve misuse case model. In Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec) (pp. 94-99). IEEE.

Al-Saqqa, S., Sawalha, S., & AbdelNabi, H. (2020). Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11).

Ardo, A. A., Bass, J. M., & Gaber, T. (2022, August). Towards secure agile software development process: A practice-based model. In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 149-156). IEEE.

Bezerra, C. M. M., Sampaio, S. C., & Marinho, M. L. (2020, August). Secure agile software development: policies and practices for agile teams. In International Conference on the Quality of Information and Communications Technology (pp. 343-357). Cham: Springer International Publishing.

Caniglia, A., Dentamaro, V., Galantucci, S., & Impedovo, D. (2025). FOBICS: Assessing project security level through a metrics framework that evaluates DevSecOps performance. Information and Software Technology, 178, 107605.

Kundi, M., & Chitchyan, R. (2014, August). Position on metrics for security in requirements engineering. In 2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET) (pp. 29-31). IEEE.

Maier, P., Ma, Z., & Bloem, R. (2017, August). Towards a secure scrum process for agile web application development. In Proceedings of the 12th International Conference on Availability, Reliability and Security (pp. 1-8).

Fazzani, F. H. (2023). A Scrum Model as a Tool for Track Changing Requirements. Bani Waleed University Journal of Humanities and Applied Sciences, 8(1), 798-811.

Mallouli, W., Cavalli, A. R., Bagnato, A., & De Oca, E. M. (2020, July). Metrics-driven DevSecOps. In ICSOFT (pp. 228-233).

Mohaddes Deylami, H., Ardekani, I., Muniyandi, R. C., & Sarrafzadeh, H. (2015). Effects of software security on software development life cycle and related security issues.

Moyon, F., Almeida, P., Riofrío, D., Mendez, D., & Kalinowski, M. (2020, August). Security compliance in agile software development: A systematic mapping study. In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 413-420). IEEE.

Nagele, S., Watzelt, J. P., & Matthes, F. (2022, June). Investigating the current state of security in large-scale agile development. In International Conference on Agile Software Development (pp. 203-219). Cham: Springer International Publishing.

Ojuri, M. A. (2024). Integrating Cybersecurity Standards into Software Quality Assurance Frameworks: A Holistic Approach. Journal of Computer Science and Technology Studies, 6(1), 258-271.

OWASP (2020), "OWASP SAMM, version 2" available at https:// owaspsamm.org/model/. Accessed in July 2025.

Mohamed, H. K., & Nabus, H. S. (2025). Developing a Software Agent to Access Drug-Related Information from the Dark Web. Bani Waleed University Journal of Humanities and Applied Sciences, 10(3), 415-435.

Rindell, K., Hyrynsalmi, S., & Leppänen, V. (2018, May). Aligning security objectives with agile software development. In Proceedings of the 19th International Conference on Agile Software Development: Companion (pp. 1-9).

Rindell, K., Ruohonen, J., Holvitie, J., Hyrynsalmi, S., & Leppänen, V. (2021). Security in agile software development: A practitioner survey. Information and Software Technology, 131, 106488.

Savola, R. M., Frühwirth, C., & Pietikäinen, A. (2012). Risk-driven security metrics in agile software development-an industrial pilot study. J. Univers. Comput. Sci., 18(12), 1679-1702.

Sherif, E., Yevseyeva, I., Basto-Fernandes, V., & Cook, A. (2024). The Smart Approach to Selecting Good Cyber Security Metrics. Journal of Internet Services and Information Security, 14(4), 312-330.

Sinha, A., & Das, P. (2021, September). Agile methodology vs. traditional waterfall SDLC: A case study on quality assurance process in software industry. In 2021 5th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech) (pp. 1-4). IEEE.

Sultan, K., En-Nouaary, A., & Hamou-Lhadj, A. (2008, April). Catalog of metrics for assessing security risks of software throughout the software development life cycle. In 2008 International Conference on Information Security and Assurance (ISA 2008) (pp. 461-465). IEEE.

Thool, A., & Brown, C. (2024, June). Securing agile: Assessing the impact of security activities on agile development. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 668-678).

Dreheeb, A. M., & El Tajouri, H. (2025). Role Of Artificial Intelligence In Enhancing Cyber Security. Bani Waleed University Journal of Humanities and Applied Sciences, 10(3), 121-129.

Vacca, J. R. (Ed.). (2012). Computer and information security handbook. Newnes.

Vicente Mohino, J., Bermejo Higuera, J., Bermejo Higuera, J. R., & Sicilia Montalvo, J. A. (2019). The application of a new secure software development life cycle (S-SDLC) with agile methodologies. Electronics, 8(11), 1218.

Wagner, T. J., & Ford, T. C. (2020, February). Metrics to meet security & privacy requirements with agile software development methods in a regulated environment. In 2020 International Conference on Computing, Networking and Communications (ICNC) (pp. 17-23). IEEE.