

Comparative Analysis of Security Testing Strategies across Various Development Methodologies

Jamaluddin Jasmis, Nurul Huda Nik Zulkipli*, Azlan Abdul Aziz, Mohd Nor Hasrol Jono, Azmi Aminordin, Zainal Fikri Zamzuri

Faculty of Computer Science and Mathematics, Universiti Teknologi MARA, Melaka, MALAYSIA

* Corresponding Author

DOI: <https://doi.org/10.47772/IJRISS.2026.100900750>

Received: 30 September 2026; Accepted: 05 October 2026; Published: 22 October 2026

ABSTRACT

The increasing sophistication of cyber threats has made software security an essential requirement throughout the Software Development Life Cycle (SDLC) rather than an activity performed only during the final stages of development. This study presents a comparative review of three widely adopted application security testing techniques—Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST)—across three software development methodologies: Waterfall, Agile, and DevSecOps. Recent literature was synthesised to evaluate the characteristics, strengths, limitations, and suitability of each testing technique within different development environments. The findings indicate that SAST and DAST remain effective in structured development processes but face integration and scalability challenges in Continuous Integration and Continuous Deployment (CI/CD) environments. In contrast, IAST provides runtime context, improves vulnerability detection accuracy, reduces false positives, and better supports continuous security testing in DevSecOps. Based on these findings, this study proposes a **Security Testing Selection Framework (STSF)** to assist practitioners in selecting appropriate security testing strategies according to development methodology, automation capability, and CI/CD integration requirements. The proposed framework translates comparative findings into a practical decision-support model that supports the adoption of secure, resilient, and continuously integrated software development practices.

INTRODUCTION

The rapid digital transformation of governments, businesses, and critical infrastructures has significantly increased dependence on software systems. As software applications become more interconnected and support essential services, they have also become attractive targets for increasingly sophisticated cyberattacks. Recent cybersecurity reports indicate that vulnerabilities introduced during software development remain one of the leading causes of successful security breaches. Consequently, software security should no longer be viewed as an activity performed only during the final testing phase but as an integral part of the entire Software Development Life Cycle (SDLC).

Over the past decade, software development methodologies have evolved from traditional sequential approaches, such as the Waterfall model, to more adaptive methodologies including Agile and, more recently, DevOps and DevSecOps. While these methodologies have improved software delivery speed, collaboration, and product quality, they have also introduced new challenges for maintaining effective security practices. Security testing strategies originally designed for sequential development processes are often difficult to integrate into modern Continuous Integration and Continuous Deployment (CI/CD) pipelines, where software changes are developed, tested, and deployed continuously.

Application security testing techniques have evolved alongside these development methodologies. Static Application Security Testing (SAST) identifies vulnerabilities by analysing source code during the early stages of development, whereas Dynamic Application Security Testing (DAST) evaluates running applications by simulating external attacks. More recently, Interactive Application Security Testing (IAST) has emerged as a hybrid approach that combines static and dynamic analysis to provide runtime context, improve vulnerability

detection accuracy, and reduce false-positive findings. Although these techniques are widely adopted, their effectiveness depends on the characteristics and requirements of the software development methodology in which they are implemented.

Despite the growing body of research on secure software development, there is still limited understanding of

how different security testing techniques should be applied across software development methodologies. Most existing studies focus on individual testing techniques or specific development approaches rather than providing a comprehensive comparison across multiple methodologies. Although the strengths and limitations of SAST, DAST, and IAST have been widely discussed, limited attention has been given to translating these findings into practical guidance that helps practitioners select appropriate security testing strategies based on software development methodology, workflow characteristics, and deployment requirements. This limitation has become increasingly important as organisations continue to adopt Agile and DevSecOps practices that rely on automated CI/CD pipelines.

To address this gap, this study presents a comparative review of three widely adopted application security testing techniques, namely Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST), across three software development methodologies: Waterfall, Agile, and DevSecOps. The review examines the characteristics, strengths, limitations, implementation challenges, and suitability of each technique throughout the SDLC. Based on the comparative analysis, this study further proposes a Security Testing Selection Framework (STSF) to support the systematic selection of security testing strategies according to software development methodology, workflow characteristics, automation capability, and CI/CD integration requirements. Rather than introducing a new security testing technique, the proposed framework translates comparative findings into a practical decision-support model that assists software practitioners in selecting security testing strategies appropriate to different software development environments.

The primary contributions of this paper are as follows:

1. To provide a comprehensive comparative analysis of SAST, DAST, and IAST across Waterfall, Agile, and DevSecOps software development methodologies.
2. To examine the strengths, limitations, implementation challenges, and CI/CD integration capability of each security testing technique throughout the SDLC.
3. To identify the technological innovation gap between conventional application security testing approaches and modern CI/CD-oriented software development environments.
4. To propose a Security Testing Selection Framework (STSF) that supports the systematic selection of security testing strategies based on software development methodology, workflow characteristics, automation capability, and deployment requirements.
5. To provide practical recommendations for integrating security testing into modern secure software development practices.

Related Work

Software security has become a critical concern as software systems increasingly support essential services across governments, businesses, and everyday activities [1][2]. Despite its importance, integrating security throughout the Software Development Life Cycle (SDLC) remains a significant challenge, particularly as organisations adopt diverse software development methodologies [2][3]. Although numerous secure software development approaches have been proposed, achieving consistent security integration across different development environments continues to be difficult. This section reviews recent studies on security testing techniques within different software development methodologies, focusing on their evolution, implementation challenges, and proposed solutions for improving secure software development.

Evolution and Challenges in Secure Software Development

Historically, security activities were often addressed during the later stages of software development, allowing vulnerabilities to remain undetected until deployment [2][4]. As cyber threats have become increasingly

sophisticated, this reactive approach has proven inadequate for protecting modern software systems [1]. Recent studies consistently suggest that integrating security throughout the Software Development Life Cycle (SDLC) is both more effective and more cost-efficient than addressing vulnerabilities after software deployment. Consequently, security has evolved from a standalone testing activity into a continuous process that spans the entire software development lifecycle.

Despite this shift, many organisations continue to encounter difficulties when adopting secure SDLC frameworks. Common challenges include implementation complexity, limited organisational resources, methodological inflexibility, and insufficient understanding of secure software development practices [3][5]. Furthermore, the widespread adoption of many proposed secure software development methodologies remains limited because empirical evidence demonstrating their effectiveness is still insufficient [3]. In addition, developers often lack practical guidance on selecting security mechanisms that can be effectively integrated into existing software development activities [5]. These challenges highlight the need for practical approaches that support the systematic integration of security throughout modern software development processes.

Security Testing in Traditional vs. Agile Methodologies

Traditional software development methodologies, particularly the Waterfall model, incorporate security activities within clearly defined SDLC phases, allowing security planning, implementation, and verification to be conducted in a structured manner [6]. This structured approach has served as the foundation for many early security requirement frameworks because security-related activities can be scheduled and managed at predetermined stages of the development process.

In contrast, Agile development emphasises rapid iterations, continuous customer feedback, and frequent software releases. Although these characteristics improve development flexibility and responsiveness, they also make the integration of comprehensive security activities more challenging [7][8]. Security practices that require extensive documentation or lengthy assessment procedures may conflict with Agile's short development cycles, increasing the risk that security activities receive less attention during sprint execution. Consequently, several studies have highlighted the need for security practices that are specifically adapted to Agile development environments rather than directly adopting approaches originally designed for traditional development models [6].

The adoption of security testing also varies across software development methodologies. Automation testing is commonly implemented in both Waterfall and Agile environments, whereas its adoption is considerably higher in DevOps environments because of the greater reliance on automation and CI/CD pipelines [9]. Although dedicated security testing activities remain relatively limited across all methodologies, they are reported more frequently in Agile environments than in traditional development approaches [9]. These findings suggest that while Agile development encourages continuous testing, additional effort is still required to integrate security testing effectively without reducing development efficiency.

Several secure software development frameworks have been proposed to address these challenges. For example, the Singapore methodology promotes security-by-design principles within Agile development by embedding security activities into iterative development cycles. While the fundamental security principles remain consistent with those applied in traditional methodologies, the framework adapts security activities to support incremental software delivery and incorporates penetration testing during transition stages to improve overall software security [3].

Security in DevOps and DevSecOps

The emergence of DevOps has fundamentally changed the way software is developed and delivered by promoting continuous integration, continuous delivery, and close collaboration between development and operations teams. As software release cycles become shorter and more frequent, security can no longer be treated as a separate activity performed only before deployment. This evolution has led to the adoption of DevSecOps, which integrates security throughout the software delivery pipeline rather than introducing it at the final stages of development [4][10].

Within a DevSecOps environment, security activities such as threat modelling, secure coding, vulnerability assessment, and dynamic security testing are incorporated into every stage of the CI/CD pipeline [4][10]. The objective is to identify and address security issues continuously while maintaining rapid software delivery. This approach enables organisations to improve software security without compromising development efficiency.

Despite these advantages, conventional security testing techniques such as Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) often struggle to satisfy the speed and automation requirements of DevOps environments. These techniques were originally designed for traditional sequential software development processes and may introduce scalability and performance challenges when integrated into automated CI/CD pipelines [11]. As a result, organisations frequently need to adapt existing security testing processes to support continuous software delivery.

Several studies recommend adopting security testing approaches that minimise disruption to software deployment while maintaining effective vulnerability detection. One commonly proposed solution is the use of parallel testing pipelines, allowing resource-intensive SAST scans to be executed without delaying software releases [12]. In addition, Application Security Testing as a Service (TaaS) has been suggested as an alternative for organisations with limited in-house security expertise, enabling access to specialised security testing capabilities without requiring significant internal resources [12].

Interactive Application Security Testing (IAST) has emerged as a hybrid security testing approach that combines the strengths of static and dynamic analysis while providing runtime context during application execution [12]. Compared with conventional testing techniques, IAST offers improved vulnerability detection accuracy and better integration with automated software delivery pipelines, making it particularly suitable for DevSecOps environments where continuous security assessment is essential.

Comparative Overview of SAST, DAST, and IAST

Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST) represent three of the most widely adopted application security testing techniques in modern software development. Although all three approaches aim to identify software vulnerabilities, they differ considerably in terms of testing methodology, execution environment, and integration within the Software Development Life Cycle (SDLC) [11].

SAST analyses application source code, bytecode, or binaries without executing the software, allowing security vulnerabilities to be identified during the early stages of development [11]. Because testing is performed before software deployment, developers can detect and remediate security weaknesses before they propagate to later development phases. However, SAST may generate false-positive results and is generally less effective in identifying vulnerabilities that depend on runtime behaviour or application configuration [11].

Unlike SAST, DAST evaluates software while it is running by simulating external attacks against deployed applications [11]. This approach is capable of identifying vulnerabilities that are only exposed during execution, including authentication weaknesses, session management issues, and configuration-related security flaws. Since DAST requires a fully operational application, security testing is typically conducted during the later stages of software development or after deployment. Although DAST provides realistic security assessments, it cannot identify vulnerabilities within the application's source code and may require longer testing durations in complex environments [11].

IAST combines the strengths of static and dynamic security testing by monitoring application behaviour during execution while simultaneously analysing interactions within the application [12]. This hybrid approach provides contextual information that improves vulnerability detection accuracy and reduces both false-positive and false-negative findings. In addition, IAST integrates effectively into automated CI/CD pipelines, making it particularly suitable for Agile and DevSecOps environments that require continuous security testing [12].

Each security testing technique offers distinct advantages depending on the software development methodology and project requirements. SAST remains well suited to traditional development environments

where early code analysis is prioritised, whereas DAST is effective for validating the security of deployed applications. IAST, in contrast, is particularly beneficial in modern software development environments that require continuous security assessment and rapid feedback throughout the software delivery process. These differences suggest that no single security testing technique is universally applicable across all software development methodologies.

Therefore, selecting an appropriate security testing strategy requires consideration of both the characteristics of the software development methodology and the operational requirements of the development environment. This observation forms the basis of the comparative analysis presented in the following section, which examines the suitability of SAST, DAST, and IAST across Waterfall, Agile, and DevSecOps methodologies.

Although previous studies have extensively examined the characteristics, strengths, and limitations of SAST, DAST, and IAST, existing research has largely focused on individual security testing techniques or specific software development methodologies. A comprehensive comparison that systematically evaluates these testing approaches across multiple development methodologies remains limited. More importantly, few studies have translated comparative findings into practical guidance for selecting appropriate security testing strategies according to software development methodology characteristics. To address this gap, the present study adopts a structured comparative review methodology to evaluate the suitability of SAST, DAST, and IAST across Waterfall, Agile, and DevSecOps environments. The methodology employed in this study is described in the following section.

RESEARCH METHODOLOGY

This study adopts a qualitative comparative literature review methodology to examine the suitability of three application security testing techniques, namely Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST), across three software development methodologies: Waterfall, Agile, and DevSecOps. The study aims to identify the strengths, limitations, implementation challenges, and suitability of each security testing technique throughout the Software Development Life Cycle (SDLC).

Research Design

The study employed a structured literature review combined with qualitative comparative analysis of published literature. Peer-reviewed journal articles and conference papers were prioritised, while selected technical reports and reputable industry publications were used where they provided directly relevant evidence on DevSecOps and CI/CD practices. The review focused on studies published between 2020 and 2025 to capture contemporary software development practices, application security testing, and emerging security technologies. The objective was to synthesise relevant evidence rather than to conduct a formal systematic review with statistical meta-analysis.

Literature Search Strategy

Relevant publications were identified through major digital libraries, including IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Google Scholar. The literature search used combinations of the following keywords:

- "Software Security Testing"
- "Static Application Security Testing (SAST)"
- "Dynamic Application Security Testing (DAST)"
- "Interactive Application Security Testing (IAST)"
- "Software Development Life Cycle (SDLC)"
- "Waterfall"
- "Agile"
- "DevOps"
- "DevSecOps"

- "CI/CD Security"

Boolean operators (AND and OR) were applied to refine the search strategy and improve the relevance of the retrieved publications.

Selection Criteria

The selected publications were considered relevant when they satisfied the following inclusion criteria:

- Published between 2020 and 2025.
- Peer-reviewed journal articles or conference papers.
- Studies focusing on software security testing techniques.
- Studies discussing the implementation of security testing within software development methodologies.
- Publications written in English.

Publications were excluded when they were duplicates, lacked sufficient technical discussion, were primarily opinion-based, or were not sufficiently related to application security testing or software development methodologies. Peer-reviewed evidence was prioritised during interpretation, while supplementary technical and preprint sources were retained only when they addressed a specific aspect of the comparison.

The review did not retain a formal PRISMA-style record of the number of records retrieved, duplicates removed, full texts screened, and studies excluded at each stage. Similarly, no formal numerical quality-assessment score was assigned to the included publications. To reduce this limitation, the synthesis emphasised source relevance, methodological clarity, direct discussion of security testing, and evidence applicable to Waterfall, Agile, DevSecOps, or CI/CD environments. These methodological constraints are acknowledged as limitations and should be addressed through a fully documented systematic review in future work.

Data Extraction and Comparative Analysis

The selected studies were analysed using a qualitative comparative approach to evaluate the characteristics and suitability of different application security testing techniques across software development methodologies.

The following information was extracted from each publication:

- Security testing technique employed.
- Development methodology adopted.
- SDLC phase of implementation.
- Testing characteristics.
- Strengths and limitations.
- Integration capability with Continuous Integration and Continuous Deployment (CI/CD).
- Suitability for Waterfall, Agile, and DevOps/DevSecOps environments.

The extracted data were synthesised into comparative matrices to facilitate systematic evaluation, identify implementation challenges, and highlight emerging best practices. The comparative findings also provided the basis for developing the proposed Security Testing Selection Framework (STSF), which supports the systematic selection of appropriate security testing strategies across different software development methodologies.

RESEARCH FRAMEWORK

The research framework consists of the following sequential phases:

1. Literature identification through digital databases.
2. Screening and selection of relevant publications based on predefined inclusion and exclusion criteria.
3. Extraction and categorisation of security testing characteristics and software development methodology attributes.

4. Comparative analysis to evaluate the suitability of SAST, DAST, and IAST across different software development methodologies and develop the proposed Security Testing Selection Framework (STSF) for practical security testing selection.

The proposed methodological workflow provides a structured and transparent basis for comparing current knowledge on application security testing across different software development methodologies. Because the review did not retain formal screening counts or a numerical quality-assessment procedure, the findings should be interpreted as a qualitative comparative synthesis rather than as statistically generalisable systematic-review results.

Figure 1 illustrates the overall research framework adopted in this study. The research begins with a structured literature search using reputable digital databases to identify publications related to application security testing techniques and software development methodologies. The retrieved publications are then screened according to predefined inclusion and exclusion criteria, followed by qualitative assessment of their relevance and technical contribution to the comparative analysis.

Following the screening process, key information—including the security testing technique, software development methodology, SDLC implementation phase, strengths, limitations, and CI/CD integration capability—is extracted and categorised for comparative analysis. The extracted data are subsequently synthesised through qualitative comparative analysis to evaluate the suitability of SAST, DAST, and IAST across Waterfall, Agile, and DevOps/DevSecOps environments. Finally, the comparative findings are synthesised to formulate practical recommendations and develop the proposed Security Testing Selection Framework (STSF), which supports the systematic selection of appropriate security testing strategies across different software development methodologies.

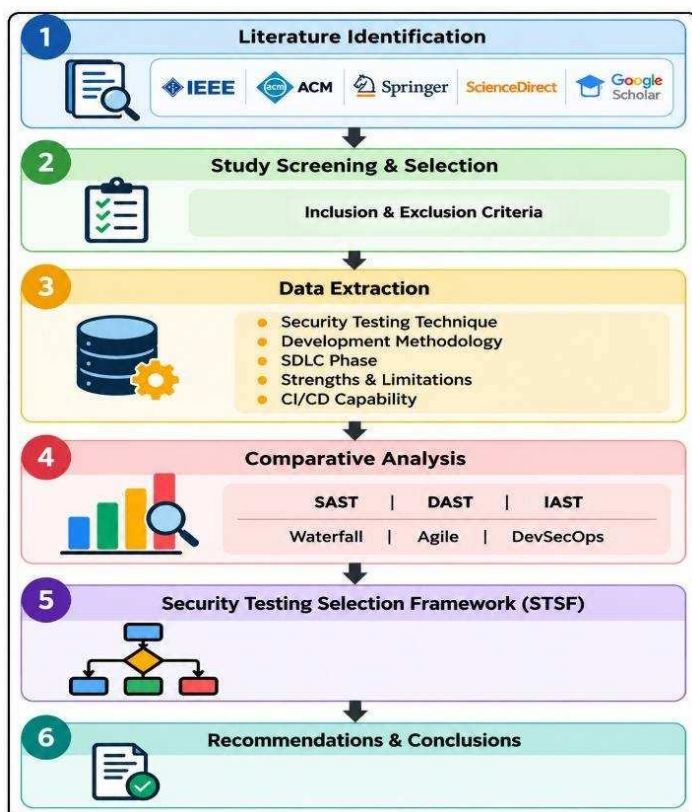


Figure 1. Research Framework for Comparative Analysis of Security Testing Strategies across Software Development Methodologies

Comparative Analysis

The integration of security testing into the SDLC has undergone a substantial transformation as development methodologies have transitioned from traditional sequential models to more iterative and continuous

approaches. The selection and execution of security testing strategies are substantially influenced by the development methodology's tempo, structure, and emphasis on automation and collaboration. Waterfall methodology is more comprehensively incorporate security activities into their structured frameworks which can accommodate for planned security assessments at specific stages [6]. However, this may lead to security aspects being treated as a separate phase rather than the continuous one. In contrast, Agile methodologies that emphasize rapid iteration and flexibility may encounter challenges in effectively incorporating security due to their immediate nature, which may result in the neglect of more time-consuming security activities. As a result, this circumstance may support the development of security protocols that are specifically designed for Agile environments [6].

The concept of DevSecOps has been required by the increase in DevOps, imposing the integration of security testing into continuous integration/continuous delivery (CI/CD) pipelines. This approach is designed to integrate security protocols, such as dynamic security testing, vulnerability assessment, secure coding techniques and threat modelling throughout the entire process [4][10]. In DevOps environments, conventional security testing approaches frequently encounter scalability issues because of their inherent limitations and their design for linear SDLCs [11]. Parallel testing pipelines and hybrid technologies have been highlighted as solutions to these challenges [12]. There are three primary security testing methodologies: SAST, DAST, and IAST. Code is evaluated by SAST without execution predominantly during the early development phase to identify potential issues [15]. In advanced phases such as penetration testing and DAST, a black-box methodology evaluates active applications to identify vulnerabilities [14] [17]. To enhance accuracy and minimize false positives and negatives, IAST, a hybrid methodology, integrates both approaches by analysing code during runtime [14][16]. Employing a combination of tools is frequently more effective than relying on a single approach, as the effectiveness of these tools can fluctuate [14]. Table 1 summarises the characteristics of SAST, DAST, and IAST and compares their suitability across different software development methodologies.

Table 1. Comparative Matrix of Security Testing Strategies

Feature	SAST	DAST	IAST
Phase of SDLC	Early development (coding, commit) [15]	Late development (testing, staging) [13]	Testing/Runtime [16]
Nature of Testing	White-box (analyzes source code/bytecode without execution) [13]	Black-box (tests running application) [13]	Hybrid (analyzes code at runtime from within the application) [14] [16]
Vulnerability Detection	Identifies code-level vulnerabilities, coding standard violations [13] [18]	Detects vulnerabilities in running application, language-independent [16]	Identifies vulnerabilities with more context, lower false positives/negatives [14] [16]
Strengths	Early detection, cost-effective for fixing issues, enforces coding standards [18]	Finds runtime vulnerabilities, programming language agnostic, good for compliance [13] [16]	Combines SAST and DAST advantages, better accuracy, context-aware [14] [16]
Limitations	High false positives, can't detect runtime issues, scalability challenges in DevOps [11]	High false positives/negatives for specific vulnerabilities, low coverage for some types [14]	Requires application to be running; agents/instrumentation must be integrated; coverage depends on exercised code paths [16]
Suitability for Waterfall	Can be integrated into specific development or testing phases [6] [16]	Suitable for dedicated testing phases before deployment [6] [16]	Less aligned with sequential phases; primary benefits best realized in iterative environments.
Suitability for Agile	Can be integrated into sprints, but may require adjustments	Applicable in testing iterations, though may not fit rapid cycles	Highly suitable due to real-time feedback and integration into continuous testing [16]

	for speed [6]	easily [7]	
Suitability for DevOps/DevSecOps	Requires parallel pipelines for deep scans, may struggle to scale [11] [12]	Can be used, but also faces scaling issues in CI/CD [11]	Particularly useful for continuous, context-aware testing, but should be combined with complementary techniques [14] [16]
Integration with CI/CD	Possible with specialized tools, might slow down pipelines [11]	Possible, but can be time-consuming for rapid deployments [11]	Can support continuous testing, subject to instrumentation, runtime coverage, and pipeline constraints [14] [16]

Security Testing Selection Framework and Practical Recommendations

The comparative analysis indicates that the suitability of SAST, DAST, and IAST is influenced by the characteristics of the adopted software development methodology. Each technique provides distinct capabilities, but its practical value depends on alignment with the development workflow, automation capability, deployment practices, application architecture, and required security coverage.

Comparative Evaluation

SAST is particularly effective for identifying code-level vulnerabilities during the early stages of software development, enabling security weaknesses to be addressed before software deployment [15]. In traditional Waterfall models, SAST can be effectively incorporated into specification, planned development or testing phases [6]. However, in Agile environments, integrating SAST into rapid development iterations can be challenging because extensive security analysis may affect sprint velocity and continuous delivery objectives [7]. Within DevOps and DevSecOps environments, SAST remains valuable but often encounters scalability challenges due to high false-positive rates and execution overhead during automated CI/CD processes [11]. Solutions like parallel testing pipelines are proposed to manage time consuming SAST scans without delaying rapid deployment [12].

DAST evaluates running applications by simulating external attacks, enabling runtime vulnerabilities that are not visible through static code analysis to be identified [13]. Independence of its programming language makes it versatile to identify issues in the final product [16]. DAST is compatible for testing phases in traditional methodologies [6]. However, the effectiveness of DAST in Agile environments may be limited because extensive runtime testing can delay rapid development iterations and continuous feedback cycles [14]. In DevOps, DAST, like SAST, faces scalability challenges as it was designed for traditional, sequential development cycles which makes it difficult to seamlessly integrate into CI/CD environments [11].

IAST represents a hybrid security testing approach that combines aspects of static and dynamic analysis by monitoring application behaviour during runtime. Its contextual view can help relate a detected weakness to the code and execution path involved, which may improve the usefulness of findings compared with approaches that analyse code or application behaviour in isolation [16]. This capability makes IAST particularly suitable for Agile and DevSecOps environments where rapid, context-aware feedback is valuable [14][16]. However, the literature does not support treating IAST as universally superior to SAST or DAST. Its effectiveness depends on application architecture, instrumentation requirements, runtime coverage, and the types of vulnerabilities being assessed. Evidence from comparative tool studies also indicates that combining complementary testing approaches can improve coverage, while the resulting false-positive and false-negative rates remain dependent on the vulnerability type and tool combination [18].

Quantitative evidence further supports a combined rather than single-technique strategy. Charoenwet et al. reported that a SAST tool generated warnings in vulnerable functions for 52% of vulnerability-contributing commits across 92 C/C++ projects; however, at least 76% of warnings in vulnerable functions were irrelevant to the contributing commits, while 22% of the commits remained undetected because of limitations in SAST rules [18]. These results illustrate both the value and the limitations of automated static analysis. Similarly, a comparative study combining SAST, DAST, and IAST tools found that multi-tool combinations could achieve

very high true-positive rates, while false-positive rates still varied across vulnerability categories [19]. Therefore, the STSF should be interpreted as a strategy-selection aid rather than evidence that one technique consistently outperforms the others.

Technological Innovation Gap

Emerging Security Technologies for DevSecOps

The scope of security testing in modern DevSecOps environments is increasingly extending beyond application-level SAST, DAST, and IAST. Software Composition Analysis (SCA) is used to identify vulnerable or outdated third-party components and can support software supply-chain security through Software Bills of Materials (SBOMs) [20]. This is particularly relevant to CI/CD environments because application security risks can originate from dependencies that are not directly visible in the application's source code. Container security scanning is similarly important for cloud-native delivery, where vulnerabilities may exist in container images, base images, orchestration configurations, or runtime environments [21]. NIST guidance for software supply-chain security in DevSecOps also recommends scanning container images for vulnerabilities before deployment and using policy-based controls to determine whether images should be admitted to an environment [22].

AI-assisted security testing is another emerging area. Recent research has examined the use of large language models for vulnerability detection, fuzzing, test generation, program repair, bug reproduction, and vulnerability triage [23]. However, current evidence also shows that LLM-based security analysis can be inconsistent and may produce incorrect or non-robust vulnerability assessments [24]. AI-assisted techniques should therefore complement, rather than replace, established security testing and human review. Taken together, SCA, container scanning, cloud-native security controls, and AI-assisted analysis indicate that future versions of STSF could be extended from application testing selection toward a broader DevSecOps security-control selection model.

One of the key findings identified in the literature is the existence of a technological innovation gap between modern software delivery practices and conventional application security testing technologies [11]. While software delivery methods have advanced significantly with the advent of DevOps, many existing application security technologies still focus on traditional software development cycles. This inconsistency leads to difficulties in performing adequate security testing in modern environments due to the limitations of these older technologies in meeting up with the requirements of rapid and continuous development [11]. This gap necessitates the adoption of newer, more adaptable tools and strategies to ensure security keeps pace with development momentum. Furthermore, the limitation of concrete guidelines for selecting and effectively integrating security mechanisms coupled with insufficient empirical evidence for many secure software development methodologies may lead to the challenges faced by organizations in adopting and implementing effective security practices [3][5]. These findings indicate that organisations require practical guidance for selecting security testing strategies that align with modern software development methodologies and automation-oriented software delivery practices.

Combined Security Testing Strategy

The comparative analysis indicates that no single security testing technique is universally suitable for every software development environment [14]. For instance, a hybrid approach integrating static and dynamic analysis is often recommended for comprehensive security testing particularly in complex web applications [16][17]. The strength of IAST lies in its ability to bridge the gap between static and dynamic analysis, enhancing the accuracy and contextual insights which thereby making it a valuable component of a layered security testing strategy across various development methodologies [16]. Consequently, adopting a layered security testing strategy that combines multiple complementary testing techniques provides broader vulnerability coverage and improves overall software security.

Table 2 summarises the comparative evaluation of SAST, DAST, and IAST across the three software development methodologies discussed in this study. The comparison highlights the strengths, implementation challenges, and overall suitability of each security testing technique according to software development

workflow, runtime requirements, contextual accuracy, and CI/CD integration capability. These findings provide the basis for developing the proposed Security Testing Selection Framework (STSF) presented in the following subsection.

Table 2. Comparative Evaluation of Security Testing Strategies Across Software Development Methodologies

Criteria	Waterfall	Agile	DevSecOps
Early Vulnerability Detection	SAST is effective in dedicated early phases [13].	SAST can be integrated, but its cumbersome nature challenges Agile's speed [7].	SAST struggles to scale; parallel pipelines needed for deep scans [11] [12].
Runtime Vulnerability Detection	DAST suitable for defined testing phases [16].	DAST applicable in iterations, but false positives/low coverage can impact rapid feedback [14].	DAST faces scalability issues and integration challenges within CI/CD [11].
Contextual Accuracy	SAST/DAST are often used in separate sequential phases.	IAST can provide runtime context and faster feedback [16].	IAST can support context-aware testing, but should complement SAST/DAST [14] [16]
Integration Challenges	Lower, as security phases are clearly defined.	Higher, as security activities can be cumbersome, conflicting with Agile's core principles [7].	Significant, due to the "technological innovation gap" of traditional tools [11].
Overall Recommendation	Combination of SAST and DAST in sequential phases [17].	SAST with lightweight runtime testing such as IAST where appropriate [16].	Layered SAST + IAST + DAST, with automation and pipeline constraints considered [12] [19]

As shown in Table 2, no single security testing technique consistently outperforms the others across all software development methodologies. Instead, the comparative evaluation demonstrates that the suitability of SAST, DAST, and IAST depends on project characteristics, development workflow, and automation requirements. These observations provide the foundation for the proposed Security Testing Selection Framework (STSF), which translates the comparative findings into practical guidance for selecting appropriate security testing strategies.

Proposed Security Testing Selection Framework (STSF)

Based on the comparative analysis presented in this study, a **Security Testing Selection Framework (STSF)** is proposed to assist software practitioners and researchers in systematically selecting appropriate application security testing strategies according to the characteristics of different software development methodologies. While previous studies have primarily compared the capabilities of Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST), they generally evaluate these techniques independently without providing practical guidance for selecting suitable testing strategies within specific development environments. Consequently, organisations often encounter difficulties in aligning security testing practices with modern software delivery workflows, particularly in Agile and DevSecOps environments where automation and continuous integration are fundamental requirements.

The proposed STSF addresses this limitation by transforming the comparative findings into a literature-informed decision-support framework. Rather than recommending a single security testing technique, the

framework evaluates project characteristics before identifying an appropriate combination of security testing strategies. Four primary decision factors are considered, namely the adopted software development methodology, development workflow characteristics, automation capability, and Continuous Integration and Continuous Deployment (CI/CD) integration requirements. These factors are intended to guide selection rather than prescribe a universally optimal configuration.

For projects adopting the Waterfall methodology, the framework recommends integrating SAST during the implementation phase and DAST during system testing. Since Waterfall follows a structured and sequential development process, security verification activities can be planned at predefined SDLC phases without significantly affecting software delivery schedules. This approach enables comprehensive code analysis and runtime vulnerability assessment while maintaining systematic security verification throughout the development lifecycle.

In Agile environments, software development is characterised by short development iterations and continuous feature enhancement. Consequently, security testing activities must minimise disruption to sprint execution while providing rapid feedback to developers. The proposed framework therefore recommends combining lightweight SAST with IAST, allowing early vulnerability detection together with runtime contextual analysis. This combination supports continuous security improvement without compromising Agile development velocity.

For DevSecOps environments, software delivery is highly dependent on automated CI/CD pipelines, where security testing must operate continuously throughout software integration and deployment. Conventional security testing approaches alone are often insufficient due to execution overhead and scalability limitations. Accordingly, the proposed Security Testing Selection Framework (STSF) translates the comparative findings into a practical decision-support model that supports the systematic selection of application security testing strategies across different software development methodologies. By considering workflow characteristics, automation capability, and CI/CD integration requirements, the framework provides software practitioners with practical guidance for improving security testing integration while supporting the adoption of secure and resilient software development practices. This layered approach enables early code analysis, runtime vulnerability assessment, and continuous validation of deployed applications while maintaining rapid software delivery.

The proposed framework further demonstrates that the effectiveness of security testing is influenced not only by the technical capability of individual testing techniques but also by their compatibility with software development workflows. The comparative analysis indicates that no single security testing approach is universally applicable across all development methodologies. Instead, organisations should adopt security testing strategies that align with project characteristics, deployment frequency, automation maturity, and operational requirements. This observation highlights the importance of selecting security testing techniques from both technical and organisational perspectives rather than relying solely on vulnerability detection capability.

Overall, the proposed Security Testing Selection Framework (STSF) translates the comparative findings into a practical decision-support model for selecting appropriate application security testing strategies across different software development methodologies. By considering workflow characteristics, automation capability, and CI/CD integration requirements, the framework provides systematic guidance for selecting suitable combinations of SAST, DAST, and IAST while supporting the implementation of adaptive and continuously integrated security testing practices in modern software development environments.

Figure 2 presents the proposed Security Testing Selection Framework (STSF) developed from the comparative analysis conducted in this study. The framework assists software practitioners in selecting appropriate application security testing strategies according to the adopted software development methodology. By considering workflow characteristics, automation capability and CI/CD integration requirements, the framework recommends suitable combinations of SAST, DAST and IAST for Waterfall, Agile and DevSecOps environments. The proposed framework serves as a practical decision-support model for improving security testing integration throughout the Software Development Life Cycle (SDLC).

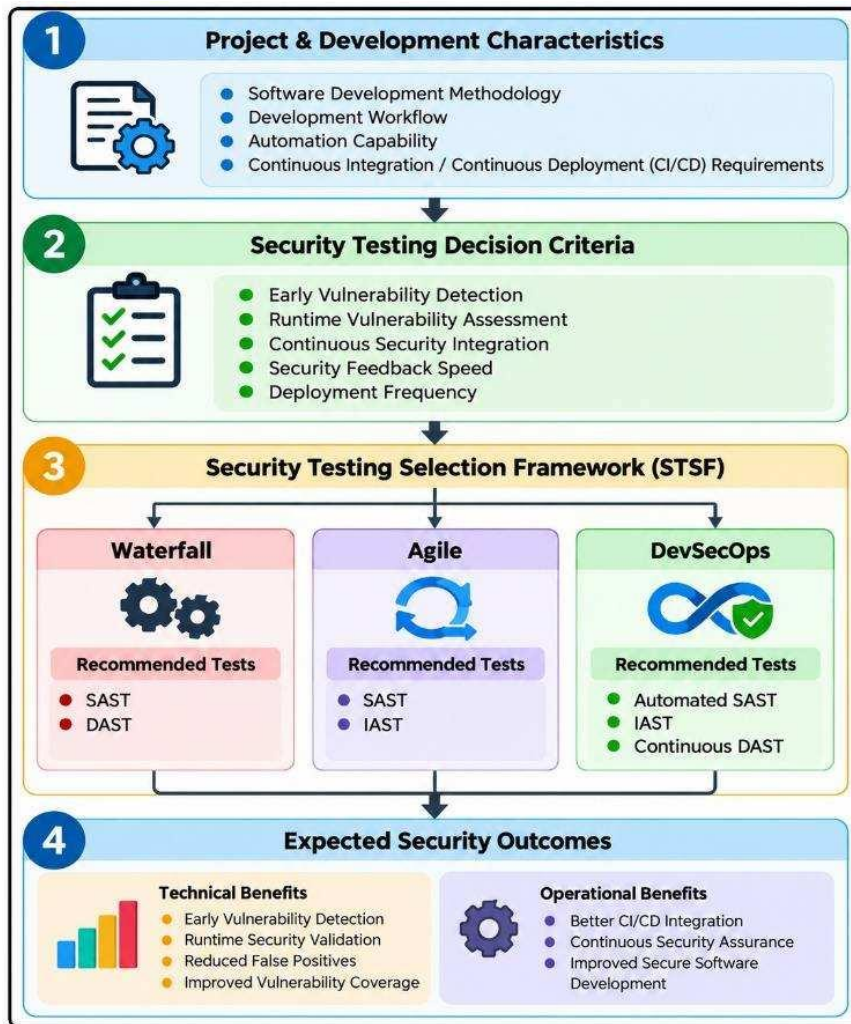


Figure 2. Proposed Security Testing Selection Framework (STSF) for Software Development Methodologies

Framework Validation Status and Practical Limitations

The STSF is a proposed framework derived from the qualitative comparison of published literature and has not yet been empirically validated. No industrial case study, controlled experiment, practitioner survey, or expert evaluation was conducted as part of the present study. Consequently, the recommendations in Table 3 should be interpreted as literature-informed guidance rather than validated prescriptions. The framework is intended to provide a structured starting point for security testing selection, while its effectiveness, usability, and generalisability require evaluation in real software development environments. Future validation should examine whether the framework improves testing coverage, vulnerability detection, false-positive management, pipeline performance, and practitioner decision-making when compared with existing security testing practices.

Table 3 presents the security testing recommendations derived from the proposed Security Testing Selection Framework (STSF). The recommendations indicate that the suitability of application security testing techniques is strongly influenced by software development methodology characteristics, including workflow structure, automation capability, and deployment practices. Waterfall environments benefit from the combination of SAST and DAST due to their structured development phases, whereas Agile environments require lightweight security testing through the integration of SAST and IAST to support rapid development iterations. For DevSecOps, the framework recommends combining automated SAST, IAST, and continuous DAST to provide continuous vulnerability detection and runtime security validation within CI/CD pipelines. These findings demonstrate that effective security testing is not determined by a single testing technique but by

selecting an appropriate combination of techniques that aligns with the operational characteristics of the adopted software development methodology.

Table 3. Security Testing Recommendations Generated by the Proposed Security Testing Selection Framework (STSF)

Development Methodology	Recommended Security Testing	Primary Benefit
Waterfall	SAST + DAST	Comprehensive security verification
Agile	SAST + IAST	Rapid security feedback
DevSecOps	Automated SAST + IAST + Continuous DAST	Continuous security assurance

CONCLUSION

This study examined the suitability of three widely adopted application security testing techniques, namely Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST), across Waterfall, Agile, and DevSecOps development methodologies. The comparative analysis indicates that security testing selection should consider not only the technical capabilities of individual techniques but also development workflow, automation maturity, deployment frequency, application architecture, and CI/CD requirements.

The findings indicate that SAST remains valuable for early code-level vulnerability detection, while DAST provides complementary assessment of vulnerabilities that become observable during application execution. IAST can provide useful runtime context and is therefore a relevant option for Agile and DevSecOps environments, but the evidence does not justify treating IAST as universally superior to SAST or DAST. Quantitative findings from prior empirical studies demonstrate that automated security testing can provide meaningful detection benefits while still producing false positives, missed vulnerabilities, and coverage limitations. A layered strategy that combines complementary techniques is therefore more defensible than reliance on a single testing approach.

Based on the comparative findings, the study proposes the Security Testing Selection Framework (STSF) as a literature-informed decision-support model for selecting combinations of security testing strategies according to development methodology, workflow characteristics, automation capability, and CI/CD integration requirements. The framework extends the comparative analysis into practical guidance while recognising that different projects may require different combinations of testing techniques.

Several limitations should be considered. First, the STSF has not been empirically validated through industrial case studies, controlled experiments, practitioner surveys, or expert evaluation. Second, the literature review did not retain formal PRISMA-style screening statistics or a numerical quality-assessment procedure, limiting the reproducibility of the selection process. Third, the analysis primarily focused on SAST, DAST, and IAST and therefore does not fully represent the broader DevSecOps security landscape. Emerging practices such as Software Composition Analysis, container security scanning, cloud-native security controls, and AI-assisted security testing should be incorporated in future extensions of the framework.

Future research should validate the STSF in real software development environments and compare its recommendations against existing security testing practices using measurable outcomes such as vulnerability detection coverage, false-positive rates, remediation effort, execution overhead, and CI/CD pipeline impact. Expert evaluation involving software security practitioners could also assess the framework's clarity, completeness, and practical utility. Such empirical validation would provide stronger evidence for the applicability and generalisability of the proposed framework.

REFERENCES

1. M. Ali, M. S. Uddin, N. Uddin, and M. M. Hasan, "Impact of Security assessment for more secure software – A Tactics and Multi-Dimensional Perspective," Research Square (Research Square), Mar. 2024, doi: 10.21203/rs.3.rs-3999692/v1.
2. M. O. Chan and S. Yazid, "A Novel Framework for Information Security During the SDLC Implementation Stage: A Systematic Literature Review," Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi), vol. 8, no. 1, p. 88, Feb. 2024, doi: 10.29207/resti.v8i1.5403.
3. A. Kudriavtseva and O. Gadyatskaya, "Secure Software Development Methodologies: A Multivocal Literature Review," arXiv (Cornell University), Dec. 2022, doi: 10.48550/arxiv.2211.16987.
5. V. Casola, A. D. Benedictis, C. Mazzocca, and V. Orbinato, "Secure software development and testing: A model-based methodology," Computers & Security, vol. 137, p. 103639, Dec. 2023, doi: 10.1016/j.cose.2023.103639.
6. Z. A. Maher, "Challenges and limitations in secure software development adoption - A qualitative analysis in Malaysian software industry prospect," Indian Journal of Science and Technology, vol. 13, no. 26, p. 2601, Jul. 2020, doi: 10.17485/ijst/v13i26.848.
7. P. K. Yeng, S. D. Wolthusen, and B. Yang, "COMPARATIVE ANALYSIS OF SOFTWARE DEVELOPMENT METHODOLOGIES FOR SECURITY REQUIREMENT ANALYSIS: TOWARDS HEALTHCARE SECURITY PRACTICE," p. 72, Apr. 2020, doi: 10.33965/is2020_2020061009.
8. A. Thool and C. Brown, "Securing Agile: Assessing the Impact of Security Activities on Agile Development," vol. 14, p. 668, Jun. 2024, doi: 10.1145/3661167.3661280.
9. D. Odera, M. Otieno, and J. E. Ounza, "Security risks in the software development lifecycle: A review," World Journal of Advanced Engineering Technology and Sciences, vol. 8, no. 2, p. 230, Apr. 09, 2023, doi: 10.30574/wjaets.2023.8.2.0101.
10. V. Stray, R. Hoda, M. Paasivaara, and P. Kruchten, Agile Processes in Software Engineering and Extreme Programming. Springer Science+Business Media, 2020. doi: 10.1007/978-3-030-49392-9.
11. X. Zhou et al., "Revisit security in the era of DevOps: An evidence-based inquiry into DevSecOps industry," IET Software, vol. 17, no. 4, p. 435, Jul. 2023, doi: 10.1049/sfw2.12132.
12. R. N. Rajapakse, M. Zahedi, and M. A. Babar, "An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps," p. 1, Oct. 2021, doi: 10.1145/3475716.3475776.
13. R. N. Rajapakse, M. Zahedi, M. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," arXiv (Cornell University). Cornell University, Mar. 15, 2021.
14. U.-S. Potti, H.-S. Huang, H.-T. Chen, and H. Sun, "Security Testing Framework for Web Applications: Benchmarking ZAP V2.12.0 and V2.13.0 by OWASP as an example," arXiv (Cornell University), Jan. 2025, doi: 10.48550/arxiv.2501.05907.
15. A. Seth, S. Bhattacharya, S. Elder, N. Zahan, and L. Williams, "Comparing Effectiveness and Efficiency of Interactive Application Security Testing (Iast) and Runtime Application Self-Protection (Rasp) Tools in A Large Java-Based System," SSRN Electronic Journal, Jan. 2022, doi: 10.2139/ssrn.4306114.
16. K. Li et al., "Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java," p. 921, Nov. 2023, doi: 10.1145/3611643.3616262.
17. R. A. Correa, J. R. B. Higuera, J. B. Higuera, J. A. S. Montalvo, M. S. Rubio, and Á. A. Magreñán, "Hybrid Security Assessment Methodology for Web Applications," Computer Modeling in Engineering & Sciences, vol. 126, no. 1, p. 89, Dec. 2020, doi: 10.32604/cmes.2021.010700.
18. A. L. S. Pupo, J. Nicolay, and E. G. Boix, "Deriving Static Security Testing from Runtime Security Protection for Web Applications," arXiv (Cornell University), vol. 6, no. 1, Jul. 2021, doi: 10.22152/programming-journal.org/2022/6/1.
19. W. Charoenwet, P. Thongtanunam, V.-T. Pham, and C. Treude, "An Empirical Study of Static Analysis Tools for Secure Code Review," Proc. ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA), pp. 691–703, Sep. 2024, doi: 10.1145/3650212.3680313.
20. F. M. Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications," Applied Sciences, vol. 10, no. 24, art. 9119, Dec. 2020, doi: 10.3390/app10249119.

21. X. Zhou, L. Cao, L. Li, Y. Wang, S. Li, and H. Liu, "A blockchain-based and microservices-architected software composition analysis system," *Journal of Software: Evolution and Process*, vol. 36, no. 10, 2024, doi: 10.1002/smr.2675.
22. D. P. V. S., S. C. Sethuraman, and M. K. Khan, "Container security: Precaution levels, mitigation strategies, and research perspectives," *Computers & Security*, vol. 135, art. 103490, Dec. 2023, doi: 10.1016/j.cose.2023.103490.
23. National Institute of Standards and Technology, "Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines," NIST Special Publication 800-204D, Feb. 2024.
24. X. Zhu, W. Zhou, Q.-L. Han, W. Ma, S. Wen, and Y. Xiang, "When software security meets large language models: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 12, no. 2, pp. 317–334, Feb. 2025, doi: 10.1109/JAS.2024.124971.
25. K. Shashwat, F. Hahn, X. Ou, D. Goldgof, L. Hall, J. Ligatti, S. R. Rajgopalan, and A. Ziaie Tabari, "A Preliminary Study on Using Large Language Models in Software Pentesting," *arXiv preprint arXiv:2401.17459*, 2024.