American Journal of Advanced Multidisciplinary Innovation and Research (AJAMIR)

E-ISSN: XXXX-XXXX ● Website: www.ajamir.com ● Email: editor@ajamir.com

Security-by-Design Practices in Rapid Application Development

Dr. Arjun Malhotra

Assistant Professor, International Institute of Information Technology Hyderabad, India

Abstract

Rapid application development has become central to contemporary software engineering because organizations increasingly rely on short release cycles, continuous integration, cloud-native services, reusable libraries, application programming interfaces, automation, and frequent product experimentation. The acceleration of delivery, however, can create security weaknesses when security activities remain concentrated near release rather than being incorporated into requirements, architecture, implementation, verification, deployment, and maintenance. This study examines the role of Security-by-Design (SbD) practices in reconciling application-delivery speed with systematic vulnerability prevention. The conceptual framework draws upon the NIST Secure Software Development Framework, OWASP Software Assurance Maturity Model, OWASP Application Security Verification Standard 5.0.0, OWASP Secure-by-Design guidance, and current Secure-by-Design principles promoted by CISA. NIST explicitly states that secure software practices can be integrated into individual software-development lifecycle implementations and are intended to reduce vulnerabilities in released software and address their root causes. Because no authentic application-development dataset was supplied, the analytical component uses a transparent simulation of 240 hypothetical rapid-development projects. Projects are differentiated according to four security-integration levels ranging from release-gate security to continuous security-by-design.

The synthetic analysis identifies a strong negative relationship between security-by-design integration and escaped high-severity vulnerabilities ($r = -0.834$), with approximately 69.6% of simulated variance in vulnerability escape associated with the integration score. Projects classified within the continuous-security condition record substantially fewer escaped high-severity vulnerabilities than projects relying primarily on late-stage controls. The analysis does not claim empirical organizational effects but demonstrates a reproducible design for future validation. The study concludes that security-by-design should not be understood as an additional approval stage imposed on rapid development. It should operate as an embedded engineering discipline combining explicit security requirements, lightweight threat modeling, secure architectural decisions, developer enablement, automated code and dependency analysis, continuous verification, secure defaults, vulnerability feedback, and measurable security outcomes.

Keywords: security-by-design; rapid application development; secure software development; DevSecOps; application security; CI/CD; secure coding; vulnerability management; software assurance

American Journal of Advanced Multidisciplinary Innovation and Research (AJAMIR)

E-ISSN: XXXX-XXXX ● Website: www.ajamir.com ● Email: editor@ajamir.com

1. Introduction

Rapid application development has altered the organization of software engineering by replacing long sequential development periods with short feedback cycles, incremental delivery, automated builds, continuous integration, reusable services, and frequent deployment. Agile, Kanban, DevOps, and related delivery models prioritize adaptability because products must respond quickly to customer requirements, operational feedback, competition, and technological change. These benefits become difficult to sustain when security is treated as a separate process undertaken only after functionality has been completed. NIST's Secure Software Development Framework directly recognizes that many software-development lifecycle models do not explicitly contain sufficient security detail and therefore recommends integrating secure development practices into whichever lifecycle an organization already uses.

The tension between security and speed does not necessarily arise because rapid development is incompatible with secure engineering. It more often arises because traditional security practices are implemented as centralized review activities that occur too late in an iterative workflow. Thool and Brown's real-world study of DAST integration in a Kanban and CI/CD environment illustrates this challenge. Their case study found that security testing can be integrated into modern delivery pipelines and reported limited disruption to normal team velocity when automation and dedicated integration work are used effectively. The study also emphasizes that rapid teams can perceive security as secondary when its tools and processes are poorly integrated with existing workflows. Security-by-design therefore requires changing where and how security decisions occur rather than simply adding more testing before release.

Contemporary security frameworks increasingly reflect this position. OWASP SAMM is deliberately technology- and process-agnostic and is intended to help organizations measure and improve security activities across the complete software lifecycle. OWASP ASVS provides developers and application owners with explicit technical requirements that can be used during design, development, testing, and procurement; its current stable release is version 5.0.0. OWASP's evolving Secure-by-Design Framework further argues that architectural and security decisions should be addressed during planning and design, with security requirements connected to implementation and verification rather than being deferred until testing. These resources collectively suggest that rapid delivery requires stronger integration of security, not less security.

Practical evidence nevertheless indicates that development teams continue to face difficulty institutionalizing security-by-design. Hermann, Peldszus, Steghöfer, and Berger interviewed 26 industry experts and found that companies generally attempt to consider security early, but specific security-by-design processes are often less systematic than formal models imply. Their participants described continuing pressure to prioritize functionality, while threat modeling and baseline security features were frequently adapted into lightweight forms compatible with existing development practices. This observation is important for rapid application development because a security framework that cannot fit within short development cycles is unlikely to be applied consistently.

The present study consequently examines security-by-design as an operational capability for rapid application development rather than as a conceptual aspiration. It proposes that security effectiveness should be evaluated through the extent to which security requirements, architecture, developer practices, testing, dependency controls, deployment safeguards, and vulnerability feedback are

incorporated into everyday delivery. Because no genuine project dataset was supplied, the quantitative component is explicitly simulation-based. A synthetic set of 240 hypothetical development projects is used to demonstrate how the relationship between security integration and vulnerability escape could be measured without presenting generated observations as real empirical evidence.

2. Objectives of the Study

2.1 To Examine Security Integration Within Rapid Development

The first objective is to examine how security-by-design can be incorporated into rapid application-development environments without transforming security into a late-stage delivery bottleneck. Particular attention is given to security requirements, architectural analysis, threat modeling, secure defaults, code review, dependency assurance, automated testing, and continuous vulnerability feedback. These practices align with NIST SSDF's principle that secure development activities should be integrated into existing lifecycle implementations rather than treated as a separate SDLC.

2.2 To Evaluate the Relationship Between Security-by-Design Maturity and Vulnerability Escape

The second objective is to use a transparent simulation to examine whether stronger security-by-design integration corresponds with lower numbers of high-severity vulnerabilities reaching production. The analysis distinguishes projects that rely primarily on release-gate security from projects implementing baseline, integrated, and continuous security practices.

2.3 To Develop an Operational Framework for Secure Rapid Delivery

The third objective is to translate established security standards and emerging empirical evidence into a practical framework that allows rapid development teams to retain delivery speed while increasing security assurance. The framework emphasizes risk-based controls, lightweight security artifacts, automation, developer ownership, security champions, measurable verification criteria, and continuous improvement. OWASP SAMM's risk-driven and iterative maturity approach is particularly relevant because it does not require organizations to adopt a single prescriptive development methodology.

3. Review of Literature

3.1 Secure Software Development and the Shift Toward Integrated Security

NIST's Secure Software Development Framework provides one of the strongest contemporary foundations for software security integration. Souppaya, Scarfone, and Dodson organize secure development around practices intended to prepare organizations, protect software, produce well-secured software, and respond to vulnerabilities. The framework is intentionally outcome-oriented and can be incorporated into different lifecycle models. NIST states that applying these practices should reduce vulnerabilities in released software, mitigate the effects of vulnerabilities that remain undetected, and help organizations address root causes so that weaknesses are less likely to recur. This orientation is compatible with rapid application development because it focuses on integrating required security outcomes rather than prescribing a slow sequential process.

OWASP SAMM complements SSDF by providing a maturity-based approach to software assurance. The framework supports the complete lifecycle, is technology and process agnostic, and is

designed to help organizations assess existing practices, define incremental improvement roadmaps, and measure security activities. For rapid-development organizations, maturity models are valuable because security improvement rarely occurs through a single large transformation. Teams can instead embed increasingly sophisticated controls across iterative development cycles.

OWASP ASVS adds a more granular verification perspective. Version 5.0.0 provides developers with requirements for secure development and supplies application owners and testers with a standardized basis for evaluating technical security controls. OWASP explicitly positions ASVS as useful for measurement, development guidance, and procurement requirements. A rapid-development team can therefore convert relevant ASVS requirements into backlog acceptance criteria or automated verification targets instead of waiting for a complete security assessment at the end of development.

CISA's Secure-by-Design initiative reinforces the organizational dimension of this approach by arguing that product security should be treated as a core business requirement and that software manufacturers should take greater responsibility for customer security outcomes. This perspective moves software security away from the assumption that customers must compensate for insecure product design after deployment.

3.2 Security-by-Design in Agile, DevOps, and Continuous Delivery

The integration of security into iterative engineering has been examined extensively because traditional security practices can conflict with rapid software delivery when they depend on sequential reviews, specialized teams, or manually intensive testing. Ashenden and colleagues observed that transitions toward Agile, continuous integration, DevOps, and DevSecOps have shortened development cycles, requiring organizations to rethink how secure software-development practices operate within everyday engineering work.

Moyón, Méndez Fernández, Beckers, and Klepper specifically examined integration of security-compliance requirements within scaled Agile development. Their S2C-SAFe framework was evaluated within Siemens' project environment and demonstrated that security compliance can be incorporated into lean and Agile engineering, although doing so introduces organizational and coordination challenges. Their work is significant because it shows that speed and security are not mutually exclusive when requirements are adapted to the operational structure of the development method.

Research concerning DevSecOps reaches a similar conclusion but identifies substantial adoption difficulties. Rajapakse and colleagues' systematic review examined challenges and proposed solutions associated with DevSecOps adoption, reinforcing the principle that security must be integrated across development and operational workflows rather than delegated exclusively to security specialists. Research on large-scale Agile environments has likewise identified numerous security approaches and highlighted the need to adapt security activities to iterative development practices rather than apply them as external controls.

Thool and Brown provide particularly useful recent field evidence. Their action-research case study integrated dynamic application security testing into an existing Kanban and CI/CD environment. The team automated DAST execution and incorporated it into its development pipeline. Participants reported that integration could be achieved with relatively limited disruption, although tool configuration, report interpretation, and cultural prioritization remained challenges. The researchers

concluded that automation was central to incorporating security scans into fast development without excessive manual intervention.

3.3 Developer Capability, Security Requirements, and Research Gap

Security-by-design ultimately depends on development decisions made before security tools execute. Hermann and colleagues' 2020 exploratory study found that industry practitioners commonly consider security during development, but security requirements are frequently sparse and formal practices are applied unevenly. Their 26 industry participants reported that threat modeling is used in practice, often in lightweight forms, and that functionality can still receive priority when resources are constrained. The same study found that ordinary developers are commonly responsible for implementing security features even though many lack detailed security knowledge, creating potential problems when libraries, configurations, or security assumptions are used incorrectly.

This evidence suggests that rapid application development requires security controls that are usable by ordinary developers. Security-by-design cannot depend entirely on continuous access to specialized security engineers. Secure defaults, reusable patterns, clearly written security requirements, automated feedback, architecture checklists, security champions, and accessible threat-modeling methods can reduce the cognitive and operational burden associated with security decisions. OWASP's Secure-by-Design Framework similarly promotes architecture reviews, explicit security requirements, data-flow and trust-boundary analysis, secure patterns, peer review, and risk triage as early lifecycle activities.

The research gap lies in connecting these practices directly with the operational objectives of rapid development. Existing studies have investigated DevSecOps, Agile security, secure coding, security testing, and software-assurance maturity, but organizations still require measurable evidence showing whether deeper security integration reduces vulnerabilities without making rapid delivery impractical. The present study addresses this gap by constructing a transparent security-integration index and illustrating how it can be associated with vulnerability-escape outcomes in a reproducible future empirical design.

4. Research Methodology

4.1 Research Design and Analytical Scope

The study employs a simulation-based quantitative research design supported by standards and empirical software-engineering literature. No actual software company, development team, source-code repository, vulnerability database, or CI/CD pipeline supplied data for the study. The quantitative analysis is therefore a methodological demonstration rather than a report of observed organizational performance.

A synthetic dataset of 240 hypothetical rapid application-development projects was generated. Each project received a Security-by-Design Integration Score ranging from 0 to 100. Higher values represent more systematic incorporation of security across requirements, architecture, implementation, verification, CI/CD, deployment, and post-release vulnerability management.

The integration continuum was divided into four analytical categories: Release-Gate Only, Baseline, Integrated, and Continuous Security-by-Design. These categories are original analytical

classifications rather than formal levels defined by NIST, OWASP, or CISA. The conceptual design nevertheless draws upon the continuous lifecycle approach advocated by NIST SSDF, OWASP SAMM, and OWASP Secure-by-Design guidance.

4.2 Proposed Measurement Instrument

A future empirical study should collect evidence from developers, architects, security engineers, DevOps personnel, product managers, repository records, security scanners, vulnerability-management systems, and deployment pipelines. The following five questionnaire items are proposed as a concise organizational screening instrument.

Table 1: Proposed Questionnaire for Security-by-Design Integration

Item	Proposed Questionnaire Statement	Primary Construct
Q1	Security requirements and misuse scenarios are documented alongside functional requirements before implementation begins.	Security requirements integration
Q2	Important architectural changes are reviewed for trust boundaries, attack paths, access-control assumptions, data exposure, and security dependencies.	Secure architecture and threat modeling
Q3	Developers receive immediate automated feedback on insecure code, vulnerable dependencies, secrets, and relevant security-policy violations during development.	Automated security feedback
Q4	Security acceptance criteria are verified within the CI/CD process before a release can progress to production.	Continuous security verification
Q5	Security defects identified after release are analyzed for root causes and used to improve requirements, coding guidance, architecture, or automated controls.	Security learning and continuous improvement

Source: Proposed by the author for future empirical validation. The questions were not administered in the present simulation.

These items should not be treated as a validated psychometric instrument in their present form. Future research should test reliability, convergent validity, discriminant validity, and relationships between questionnaire responses and objective repository or vulnerability-management metrics.

4.3 Synthetic Data Generation and Statistical Procedure

The Security-by-Design Integration Score was generated across a broad 0–100 range representing heterogeneous security maturity. The primary outcome was escaped high-severity vulnerabilities per 100

releases, defined conceptually as significant application vulnerabilities that were not identified or resolved before deployment.

Random variation was added to represent differences among development environments. The simulation assumed that stronger integration would generally correspond with lower vulnerability escape while allowing substantial project-level variability.

Pearson correlation and simple linear regression were used to evaluate the relationship. The simulation generated a correlation of approximately $r = -0.834$. The regression slope was approximately -0.119, meaning that within the constructed model, a ten-point increase in security integration corresponded with approximately 1.19 fewer escaped high-severity vulnerabilities per 100 releases.

Because all numerical observations are synthetic, inferential outputs do not establish causal effects in real software-development organizations. They demonstrate how an empirical study could be designed and analyzed.

5. Analysis

5.1 Security Integration Across Rapid-Development Projects

The overall mean Security-by-Design Integration Score in the synthetic dataset was approximately 61.21, while the overall mean number of escaped high-severity vulnerabilities was approximately 7.44 per 100 releases.

Projects in the Release-Gate Only category displayed the weakest security integration, with a mean integration score of approximately 24.83 and an average of 11.28 escaped vulnerabilities per 100 releases. Baseline projects achieved a mean integration score of 46.17 and approximately 9.51 escaped vulnerabilities.

The Integrated category showed a substantial reduction in the simulated defect-escape measure, recording approximately 6.85 vulnerabilities per 100 releases at a mean integration score of 65.49. Continuous Security-by-Design projects recorded the strongest simulated outcome, with a mean security-integration score of 86.21 and approximately 4.47 escaped high-severity vulnerabilities per 100 releases.

Table 2: Simulated Security Outcomes by Security-by-Design Integration Level

Security Integration Level	Simulated n	Mean Integration Score	Mean Escaped High-Severity Vulnerabilities per 100 Releases	SD
Release-Gate Only	25	24.83	11.28	1.57
Baseline	67	46.17	9.51	1.97
Integrated	86	65.49	6.85	1.61
Continuous Security-by-Design	62	86.21	4.47	2.03

American Journal of Advanced Multidisciplinary Innovation and Research (AJAMIR)

E-ISSN: XXXX-XXXX ● Website: www.ajamir.com ● Email: editor@ajamir.com

Source: Author-generated synthetic dataset; total simulated n = 240. These values do not represent observed software projects.

The difference between the lowest and highest integration categories is approximately 6.81 escaped high-severity vulnerabilities per 100 releases. Within the synthetic framework, continuous security integration therefore corresponds with approximately 60% fewer escaped vulnerabilities than the release-gate condition. This percentage is illustrative and must not be presented as a real-world industry estimate.

5.2 Relationship Between Security-by-Design and Vulnerability Escape

Pearson correlation analysis generated:

$r = -0.834$

The resulting coefficient of determination was approximately:

$R^2 = 0.696$

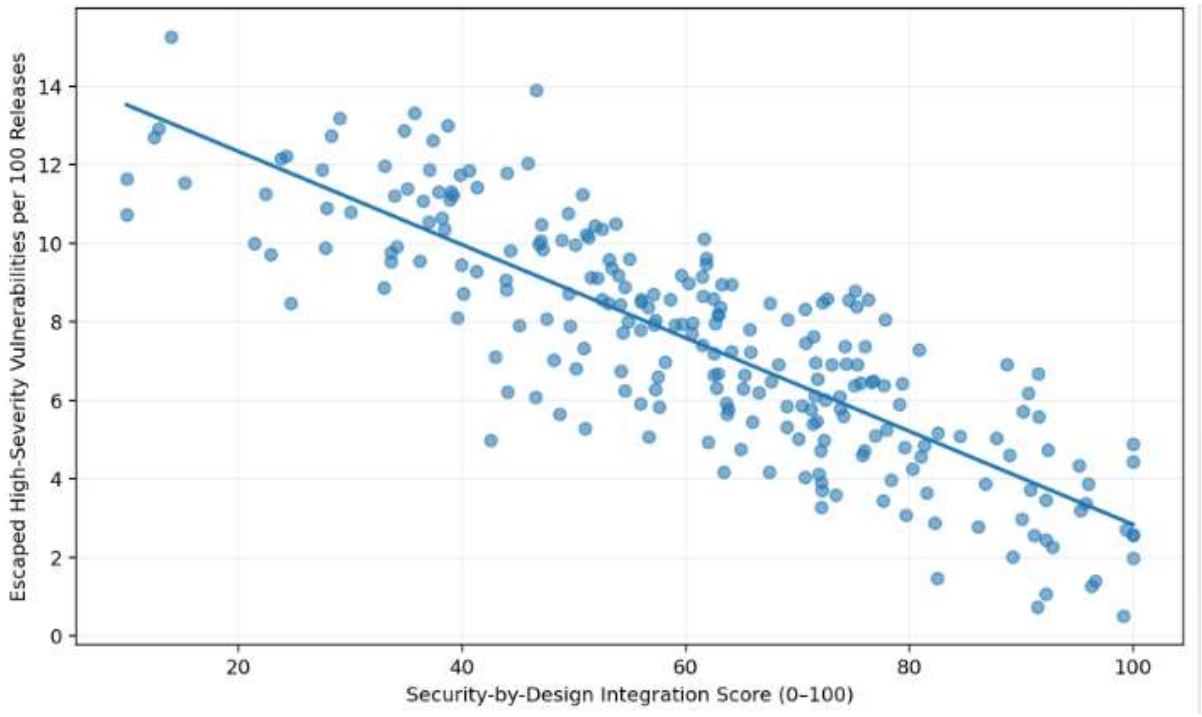
Thus, approximately 69.6% of variance in the synthetic vulnerability-escape outcome is statistically associated with variation in the Security-by-Design Integration Score.

The estimated regression relationship was approximately:

Escaped Vulnerabilities = 14.71 − 0.119(Security-by-Design Integration Score)

The negative coefficient illustrates the principal simulation assumption: greater integration of security activities is associated with progressively fewer serious vulnerabilities reaching production.

Graph 1: Simulated Relationship Between Security-by-Design Integration and Vulnerability Escape



Source: Author-generated synthetic analysis based on 240 hypothetical rapid application-development projects.

Interpretation: The downward regression line shows a strong negative simulated association between the depth of security-by-design integration and the number of high-severity vulnerabilities escaping into released software.

Key Finding: Security improvement in the simulation is associated with continuous lifecycle integration rather than dependence on final-stage security review.

5.3 Implications for Rapid Delivery Performance

The analytical result does not imply that every additional security activity automatically improves an application. Controls that generate excessive false positives, require lengthy manual review, or duplicate existing verification may slow delivery without materially reducing risk. The purpose of security-by-design is therefore not maximum security activity but appropriate security activity at the most effective point in the development workflow.

This interpretation is supported by real-world DAST integration research. Thool and Brown found that automated security testing could be introduced into Kanban and CI/CD with limited disruption when properly integrated, while also observing difficulties with scan configuration, report interpretation, and organizational prioritization. Security automation therefore creates the greatest value when results are timely, understandable, actionable, and connected directly to the work of the development team.

The finding is also consistent with Hermann and colleagues' observation that practitioners frequently use lightweight threat modeling and incorporate security features alongside functional development rather than relying exclusively on formal security processes. Rapid application development appears to require security mechanisms that preserve short feedback loops rather than introducing long handoffs between developers and centralized security teams.

6. Discussion

6.1 Security-by-Design Should Operate as Development Infrastructure

The central implication of the study is that security-by-design should not be implemented as another release approval process. When security is postponed until features are completed, weaknesses discovered during penetration testing or final security assessment may require redesign of architecture, authorization logic, APIs, data handling, or deployment configuration. NIST SSDF addresses this problem by integrating security practices into the software-development lifecycle and by emphasizing root-cause reduction rather than dependence upon post-development detection. OWASP's Secure-by-Design Framework similarly emphasizes planning, architectural decision-making, security requirements, design review, and risk triage before implementation becomes expensive to change.

In rapid development, these principles should become ordinary engineering infrastructure. Security requirements can be written into backlog items. Threat-modeling discussions can accompany substantial architectural changes. Secure design patterns can be supplied through reusable platform components. Static analysis, secret detection, dependency scanning, software-composition analysis, and

selected dynamic tests can run automatically. ASVS requirements can be translated into acceptance criteria appropriate to the application's risk profile. OWASP describes ASVS precisely as a basis for technical security verification and as guidance for developers concerning what controls should be built.
The operational objective is to shorten the security-feedback cycle. A vulnerability identified while a developer is working on a change is generally easier to understand and correct than the same weakness discovered several releases later. Rapid delivery and early security feedback therefore reinforce one another when controls are properly designed.

6.2 Automation Must Be Combined With Human Security Judgment

Automation is central to secure rapid application development but cannot replace engineering judgment. Static-analysis systems may detect suspicious source-code patterns, software-composition tools may identify vulnerable dependencies, and dynamic scanners may identify exploitable application behavior. These techniques operate at different layers and can produce incomplete or noisy results if they are used without context.

Thool and Brown's case study illustrates both sides of automation. Their organization successfully integrated DAST into CI/CD and participants generally perceived limited disruption, yet developers still reported challenges in interpreting generated reports and integrating tool output effectively into normal work. The implication is that security automation should minimize manual effort while preserving meaningful review for high-risk findings and architectural questions.

Human judgment is especially important during design. Hermann and colleagues found threat modeling to be widely used among their interview participants, although often in lightweight rather than heavily formalized forms. Their evidence also shows that developers may lack detailed security knowledge even when they are responsible for implementing security features. This combination supports practical mechanisms such as security champions, architecture consultation, approved security libraries, secure defaults, and risk-based escalation to specialist security engineers.

A mature organization therefore avoids two extremes: manual security review of every routine change and unrestricted automated approval of every security-sensitive change. Low-risk controls can be automated, while decisions involving sensitive data, new trust boundaries, authentication architecture, cryptography, privileged access, public interfaces, or high-impact dependencies can trigger additional human analysis.

6.3 Security Metrics Must Reflect Outcomes Rather Than Tool Activity

Rapid-development organizations frequently possess large quantities of security data without necessarily possessing useful security metrics. Scanner counts, number of tests executed, training completion, or number of security tickets do not independently establish that released software is becoming safer.

The simulation uses escaped high-severity vulnerabilities because it focuses on an outcome rather than a security activity. Other future empirical measures could include time to remediate critical findings, recurrence of vulnerability classes, percentage of security requirements verified automatically, percentage of critical dependencies with known ownership, mean age of unresolved vulnerabilities, exploitability-weighted vulnerability exposure, and frequency of post-release security rework.

SSSSHermann and colleagues explicitly identify the lack of effective security metrics as a practical limitation and argue that actionable measurement could improve prioritization and stakeholder understanding. OWASP SAMM similarly emphasizes measurable improvement of software-assurance activities across organizational maturity.

Security metrics in rapid development should therefore answer whether the organization is preventing important weaknesses earlier, responding faster, learning from recurring defects, and achieving stronger security without imposing disproportionate friction on delivery. The best security-by-design program is not the program that performs the largest number of controls. It is the one that consistently reduces meaningful software risk while remaining sustainable within the development process.

7. Conclusion

Rapid application development and software security are sometimes presented as competing objectives because rapid delivery favors short development cycles whereas traditional security programs can involve lengthy reviews and centralized approvals. The evidence considered in this study suggests that this conflict is largely a consequence of how security is organized. NIST SSDF, OWASP SAMM, OWASP ASVS, Secure-by-Design guidance, and empirical DevSecOps research all support embedding security into existing software-development workflows rather than postponing security evaluation until the end of the lifecycle.

The synthetic analysis illustrates the potential effect of this integration. Across 240 hypothetical rapid-development projects, the Security-by-Design Integration Score showed a strong negative relationship with escaped high-severity vulnerabilities ($r = -0.834$). Projects classified as Release-Gate Only recorded approximately 11.28 escaped high-severity vulnerabilities per 100 releases, compared with approximately 4.47 among projects represented in the Continuous Security-by-Design condition. These results are simulated and cannot be generalized as empirical industry estimates, but they demonstrate a clear quantitative framework for future organizational research.

For practice, security-by-design should begin with explicit and testable security requirements and continue through architecture, implementation, verification, deployment, and vulnerability learning. Lightweight threat modeling should accompany material design changes. Developers should receive secure defaults, approved components, actionable guidance, and rapid automated feedback. CI/CD pipelines should enforce proportionate security acceptance criteria, while specialist human review should focus on high-consequence architecture and risk decisions. Security defects identified after deployment should contribute to root-cause analysis and improvement of the development system rather than being treated as isolated defects.

Future research should validate the proposed model using genuine repository, pipeline, vulnerability, and deployment data from Agile, Kanban, DevOps, and rapid-application-development environments. Longitudinal studies would be particularly valuable because they could determine whether increasing security-by-design maturity reduces vulnerability recurrence, security rework, remediation time, and production exposure without materially degrading deployment frequency or lead time. The most sustainable model is therefore not security added to rapid development but rapid

development in which security has become an ordinary property of requirements, architecture, code, automation, and continuous engineering judgment.

References

1. Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE, 63*(9), 1278–1308. https://doi.org/10.1109/PROC.1975.9939
2. Howard, M., & Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press.
3. McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley.
4. Viega, J., & McGraw, G. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley.
5. Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley.
6. Kohnfelder, L. M. (2021). *Designing Secure Software: A Guide for Developers*. No Starch Press.
7. NIST. (2021). *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. NIST Special Publication 800-218. https://doi.org/10.6028/NIST.SP.800-218
8. Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61–72. https://doi.org/10.1109/2.59
9. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*.
10. Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software, 123*, 176–189. https://doi.org/10.1016/j.jss.2015.06.063
11. Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. *Proceedings of the International Conference on Software Process Improvement and Capability Determination*, 17–29.
12. Rahman, A. A., & Williams, L. (2016). Software security in DevOps: Synthesizing practitioners' perceptions and practices. *Proceedings of the 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery*, 70–76. https://doi.org/10.1145/2896941.2896946
13. Sharma, T., & Spinellis, D. (2018). A survey of static program analysis techniques for security. *ACM Computing Surveys, 51*(4), 1–38.
14. Viega, J. (2005). Building security in: The Microsoft SDL. *IEEE Security & Privacy, 3*(3), 55–57. https://doi.org/10.1109/MSP.2005.71
15. Arce, I., McGraw, G., & others. (2004). *Building Security In Maturity Model (BSIMM)*. Cigital.
16. Open Web Application Security Project (OWASP). (2021). *OWASP Application Security Verification Standard (ASVS) 4.0.3*. OWASP Foundation.
17. Open Web Application Security Project (OWASP). (2021). *OWASP Top 10: The Ten Most Critical Web Application Security Risks*. OWASP Foundation.
18. ISO/IEC. (2011). *ISO/IEC 25010:2011 Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*. International Organization for Standardization.

19. ISO/IEC. (2018). *ISO/IEC 27034-1:2011 Information Technology—Security Techniques—Application Security—Part 1: Overview and Concepts*. International Organization for Standardization.
20. Manadhata, P. K., & Wing, J. M. (2011). An attack surface metric. *IEEE Transactions on Software Engineering, 37*(3), 371–386. https://doi.org/10.1109/TSE.2010.60