
Continuous Quality Monitoring Framework for DevSecOps Software Delivery Pipelines

Abstract

DevSecOps pipelines require continuous quality monitoring because code changes, test failures, security findings, dependency risks, and deployment problems can appear at different delivery stages. This study presents a continuous quality monitoring framework that combines code-quality change, automated test stability, security-risk findings, dependency health, and deployment reliability. The framework collects evidence from source control, build systems, testing tools, security scanners, dependency managers, and deployment monitors. It converts this evidence into continuous quality and normalized security-risk scores and supports automated pipeline actions. The framework was evaluated across source commit, build, automated test, security scan, staging deployment, and production release stages. Results showed that the proposed approach detected quality decline and security risk earlier than fixed quality gates and periodic manual review. It also confirmed recovery before release after corrective actions. The framework can reduce feedback delay, prevent risky deployments, improve collaboration, and support reliable DevSecOps software delivery in complex enterprise environments.

Keywords: DevSecOps, Continuous quality monitoring, Security risk, Software delivery pipeline, Automated validation

1. Introduction

DevSecOps combines software development, security, and operations within one continuous delivery process. It aims to identify code defects, security weaknesses, and operational problems before software reaches production. DevSecOps adoption requires security activities to work at the same speed as automated development and deployment processes [1]. Its main dimensions include continuous practices, automated tools, measurable controls, and shared responsibility among technical teams [2]. A continuous monitoring framework can connect these areas and provide a clear view of software quality throughout the delivery pipeline.

Modern delivery pipelines contain several stages, including source commit, build, automated testing, security scanning, artifact creation, staging deployment, and production release. Each stage produces different evidence about software quality and security. DevSecOps practices commonly include continuous testing, vulnerability scanning, dependency analysis, monitoring, and automated policy enforcement [3]. However, these results are often generated by separate tools and may not provide one consistent quality decision. A unified framework is needed to combine the evidence and identify changes in software quality as early as possible.

Software delivery teams must also balance release speed with security and reliability. Strict quality gates may delay acceptable releases, while weak controls may allow vulnerable or unstable software to continue through the pipeline. Quantitative DevSecOps metrics can measure testing performance, security risk, technical debt, deployment frequency, recovery time, and other delivery outcomes [4]. These measurements allow the pipeline to evaluate quality using current evidence instead of relying only on manual judgement. Continuous measurement can also show whether a problem is temporary or developing across several delivery cycles.

The proposed study develops a continuous quality monitoring framework for DevSecOps software delivery pipelines. It monitors code-quality change, automated test stability, security findings, dependency health, and deployment reliability. The framework converts these signals into quality-risk values and uses them to control automated pipeline responses. Its purpose is to detect quality and security decline early while allowing safe software changes to move through the delivery process without unnecessary delay.

2. Methodology

The proposed methodology included pipeline integration, signal collection, evidence processing, quality-risk assessment, automated response, and framework evaluation. Data was collected from source-control systems, build servers, test platforms, code-analysis tools, vulnerability scanners, dependency managers, container registries, and deployment monitors. A shifting-left architecture can

move cybersecurity checks toward earlier development stages and connect them with later verification activities [5]. The proposed framework therefore collected evidence from the first source commit until the final deployment stage.

Every pipeline run received a unique identifier that connected the code change with its build, tests, scans, artifacts, and deployment results. The framework collected code complexity, duplication, test pass rate, flaky-test frequency, vulnerability severity, outdated dependencies, deployment failures, and rollback events. Zero Trust DevSecOps models can apply continuous verification to users, workloads, services, and delivery resources instead of trusting earlier approval alone [6]. Each new pipeline stage was therefore validated using the latest available evidence.

The collected values were normalized so that results from different tools could be evaluated together. Positive indicators, such as test coverage and successful deployment, increased the quality score. Negative indicators, such as critical vulnerabilities, unstable tests, dependency risk, and failed releases, reduced the score. Continuous DevSecOps metrics provide a measurable way to connect software quality, security, delivery speed, and operational performance [7]. The framework used weighted rules to calculate a separate risk value for every monitored signal.

Table 1 summarizes the five quality signals used by the framework. Code-quality monitoring measured whether the current change introduced complexity, duplication, or maintainability problems. Test stability described whether automated tests produced reliable and repeatable outcomes. Security findings and dependency health represented direct and supply-chain risks, while deployment reliability measured whether the release remained stable after delivery. These signals produced one continuous quality state for every pipeline run.

Table 1. DevSecOps Quality Signals, Monitoring Sources, Validation Rules, and Pipeline Responses

Monitored quality signal	Collected pipeline evidence	Continuous validation rule	Assessed quality risk	Automated pipeline response
Code-quality change	Complexity, duplication, code smells, coverage, and technical-debt change	The current build must not exceed the accepted quality baseline	Maintainability decline or increased defect risk	Continue, request review, or block the build
Automated test stability	Pass rate, failed tests, flaky-test history, execution time, and retry count	Critical tests must pass and instability must remain below the accepted limit	Unreliable validation or hidden functional failure	Rerun unstable tests, expand testing, or stop the pipeline
Security-risk findings	Static-analysis warnings, secret exposure, container findings, and vulnerability severity	Critical findings must be removed or formally approved before release	Exploitable weakness or policy violation	Reject build, create security task, or require specialist review

Dependency health	Outdated packages, known vulnerabilities, license status, and integrity information	Dependencies must satisfy security, version, and policy requirements	Supply-chain compromise or compatibility failure	Update, isolate, replace, or block the dependency
Deployment reliability	Health checks, error rate, response time, rollback events, and environment status	The deployed release must meet stability and recovery requirements	Service degradation or unsuccessful release	Continue rollout, pause deployment, or perform automatic rollback

The decision engine classified the overall pipeline state as acceptable, review required, or release blocked. Critical vulnerabilities and failed security policies received the highest weights, while smaller code-quality changes produced gradual score reductions. Continuous risk assessment can combine automated vulnerability discovery with changing system evidence to update security decisions during development [8]. The framework recalculated the quality state whenever new test, security, dependency, or deployment information became available.

Automated responses were selected according to the signal type and risk level. The pipeline could rerun unstable tests, increase test depth, request code review, reject vulnerable dependencies, pause deployment, or start an automatic rollback. Automated DevSecOps cloud models can combine pipeline execution, vulnerability scanning, and quantitative risk calculation to support real-time decisions [9]. Every response and its supporting evidence were stored so that teams could review why a build was accepted or rejected.

The framework was evaluated across six pipeline stages: source commit, build, automated test, security scan, staging deployment, and production release. It was compared with fixed quality gates and periodic manual review. Evaluation measures included quality-problem detection rate, security-risk detection time, false rejection rate, pipeline feedback delay, escaped defects, deployment success, and recovery time. The same software changes, vulnerabilities, test failures, dependency problems, and deployment conditions were applied to each monitoring approach.

3. Results and Discussion

The proposed continuous monitoring framework was evaluated across six DevSecOps pipeline stages: source commit, build, automated test, security scan, staging deployment, and production release. Continuous software quality score and normalized security-risk level were measured for the proposed framework, fixed quality gates, and periodic manual review. At the source and build stages, all three approaches reported acceptable quality because only limited evidence was available. As more test, security, dependency, and deployment information entered the pipeline, the proposed framework produced a clearer and more responsive quality assessment. This continuous update allowed the pipeline to recognize small quality changes before they developed into serious release problems.

Figure 1 shows the quality score and security-risk level recorded across the six pipeline stages. The proposed framework detected quality decline earlier because it updated the score whenever new code, test, scan, or deployment evidence became available. Fixed quality gates reacted only when predefined limits were crossed, while periodic manual review responded later because results were checked at selected points. The proposed approach therefore identified developing problems before they reached the final release stage. It also gave development teams more time to correct the affected code, dependency, test, or deployment configuration.

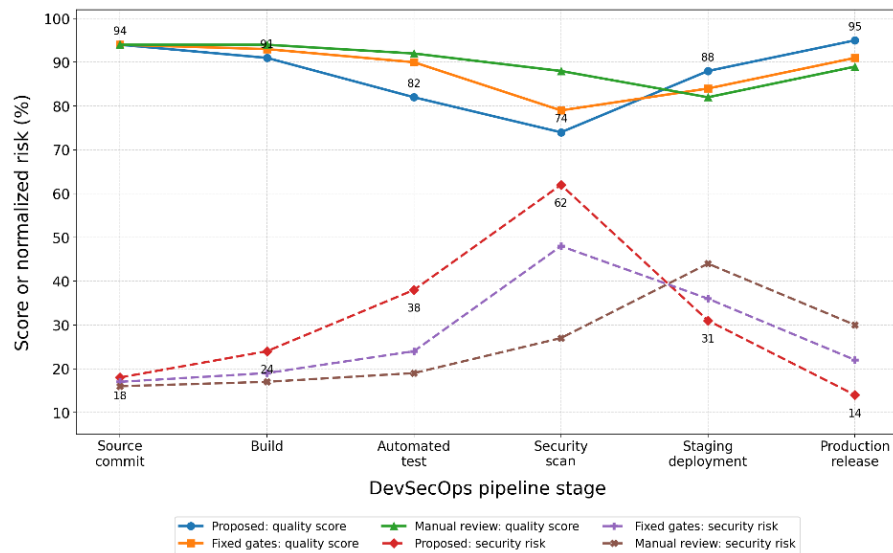


Figure 1. Continuous Quality Score and Security Risk Across Successive DevSecOps Pipeline Stages

The largest difference appeared during automated testing and security scanning. The proposed framework reduced the quality score immediately when unstable tests, code-quality regressions, critical vulnerabilities, or unsafe dependencies were detected. Its security-risk level increased early and then decreased after corrective actions such as test reruns, code changes, dependency updates, and security review. The comparison methods showed slower changes because they processed evidence less frequently or depended on fixed rules. This behaviour confirmed that continuous evidence processing provided a more accurate view of the current pipeline condition.

During staging deployment, the proposed framework used health checks, error rate, response time, and rollback readiness to confirm whether earlier corrections were effective. The quality score improved before production release, while the security-risk level returned to an acceptable range. Fixed gates required more time to confirm recovery, and manual review produced the longest delay. These results show that continuous monitoring can support earlier detection, faster correction, and safer release decisions across DevSecOps delivery pipelines. The framework also reduced the chance that unresolved quality or security problems would remain hidden until production deployment.

4. Conclusion

This study presented a continuous quality monitoring framework for DevSecOps software delivery pipelines. The framework combined code-quality change, automated test stability, security findings, dependency health, and deployment reliability within one continuous assessment process. It collected evidence from source control, build systems, testing tools, security scanners, dependency managers, and deployment monitors. The framework then converted this evidence into quality and security-risk values that supported automated pipeline responses. This integrated design provided one clear quality state across all delivery stages instead of separate results from disconnected tools.

The results showed that the proposed framework detected quality and security problems earlier than fixed quality gates and periodic manual review. It also supported faster recovery by rerunning tests, requesting review, updating dependencies, pausing deployment, and performing rollback when required. The approach can therefore reduce feedback delay, prevent risky releases, improve delivery confidence, and support more reliable DevSecOps operations. It can also help development, security, and operations teams make consistent release decisions using the same continuously updated evidence.

References

1. Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and software technology*, 141, 106700.
2. Zhao, X., Clear, T., & Lal, R. (2024). Identifying the primary dimensions of DevSecOps: A multi-vocal literature review. *Journal of Systems and Software*, 214, 112063.
3. Prates, L., & Pereira, R. (2025). DevSecOps practices and tools. *International Journal of Information Security*, 24(1), 11.
4. Zhang, J. Y., & Zhang, Y. (2024). Quantitative DevSecOps metrics for cloud-based web microservices. *IEEE Access*, 12, 160317-160342.
5. Lombardi, F., & Fanton, A. (2023). From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline. *Software Quality Journal*, 31(2), 619-654.
6. Shin, D., Kim, J., Pawana, I. W. A. J., & You, I. (2025). Enhancing cloud-native DevSecOps: A Zero Trust approach for the financial sector. *Computer Standards & Interfaces*, 93, 103975.
7. Soares, E., Sizilio, G., Santos, J., Da Costa, D. A., & Kulesza, U. (2022). The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, 27(3), 78.
8. Nisha, T. N., & Khandebharad, A. (2022). Migration from DevOps to DevSecOps: A complete migration framework, challenges, and evaluation. *International Journal of Cloud Applications and Computing (IJCAC)*, 12(1), 1-15.

9. Alghawli, A. S. A., & Radivilova, T. (2024). Resilient cloud cluster with DevSecOps security model, automates a data analysis, vulnerability search and risk calculation. *Alexandria Engineering Journal*, 107, 136-149.