International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 5, Issue 1, 126-133, 2024
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V5I1P115

Original Article

Threat Modeling Integration in DevSecOps Pipelines: Early-Stage Security Risk Identification Using Shift-Left Approaches

Pruthvi Raj Seknametla
Individual Researcher, USA.

Abstract - *Threat modeling has long been recognized as one of the highest-leverage activities in security engineering, yet it has historically been practiced as a periodic, document-heavy exercise disconnected from the pace of modern software development. This paper examines how threat modeling can be restructured architecturally and culturally to integrate continuously into DevSecOps pipelines without imposing the overhead that has traditionally made it impractical for fast moving engineering teams. Drawing on a structured study of ten organizations across seventeen months from March 2023 through July 2024, we analyze three distinct integration models and their measurable impact on early-stage risk identification rates, threat model coverage, architectural decision quality, and the cost differential between pre-development and post-deployment risk remediation. Results demonstrate that organizations with continuous threat modeling integration identify architectural security risks an average of 6.2 times earlier in the development lifecycle than those following episodic practices, and that integration significantly reduces the proportion of design-level vulnerabilities discovered in production. We also address the practical barriers to pipeline-integrated threat modeling tooling gaps, expertise distribution, and workflow friction and propose a maturity model for adoption. The findings offer concrete guidance for security architects and DevSecOps practitioners evaluating how to evolve threat modeling from a compliance activity into an engineering practice.*

Keywords - *Threat Modeling, Devsecops, Shift-Left Security, STRIDE, Attack Trees, LINDDUN, CI/CD Integration, Security by Design, DREAD, Threat-Driven Development, Architectural Risk Analysis, Security Requirements.*

1. Introduction

Walk into most security retrospectives and ask where the hardest-to-fix vulnerabilities came from, and you will hear the same answer with slight variations: the design. Not the implementation, not the dependency chain, not the configuration the fundamental architecture of the system. An authentication scheme that trusted client-supplied identifiers. A data flow that passed sensitive values through a logging subsystem. An API gateway that was designed to aggregate responses from multiple internal services without considering what an attacker could infer from the aggregate. These are not bugs that a scanner can find. They are structural choices that become vulnerabilities, and they are nearly impossible to remediate without rearchitecting the affected component. Threat modeling is the practice of systematically identifying those structural risks before they are built into a system. The concept is straightforward. Before you build something, you reason about what could go wrong, who might try to make it go wrong, and what the consequences would be. Then you design accordingly. Adam Shostack, whose 2014 book remains one of the clearest practical treatments of the subject, frames it around four questions: what are we building, what can go wrong, what are we going to do about it, and did we do a good job? It is almost embarrassingly simple as a framework. The difficulty lies in executing it consistently, at speed, across a large and constantly changing software system. That difficulty is precisely why threat modeling, despite broad acknowledgment of its value, has struggled to take hold in engineering organizations operating at DevOps velocity.

Traditional threat modeling practices were designed for waterfall-era development: a team convenes, works through a data flow diagram with a whiteboard, documents a set of threats, assigns mitigations, and produces a report that gets filed somewhere. When that report is complete, the team moves on. The application continues to evolve. Six months later, the threat model is already partially outdated, and nobody has updated it. The shift-left movement in security has drawn attention back to threat modeling by raising a simple question: if we can shift vulnerability scanning, compliance checks, and secrets detection into the pipeline, why can’t we shift threat modeling there too? The answer is that we can, partially. The challenge is that threat modeling involves a kind of structured reasoning about adversarial intent that does not compress into a static analysis scan. But it does compress into reviewable artifacts, automated validation of threat model completeness, and pipeline-integrated feedback that prompts security thinking at the moment architectural decisions are being made not weeks or months later. This paper examines how ten organizations approached this problem across a seventeen-month study period. We are interested in what integration models actually look like in practice, which ones produce the clearest security improvements, and where the friction comes from when organizations try to make threat modeling a continuous engineering activity rather than a periodic compliance exercise.

1.1. The Cost Calculus of Late Threat Discovery

The economic argument for early threat modeling is the same as the broader argument for shift-left security, but with a specific twist. Vulnerability scanners catch implementation defects insecure library versions, dangerous function calls, and misconfigured permissions. These are real risks and catching them early genuinely reduces remediation cost. But the risk of threat modeling addresses architectural design choices that create exploitable conditions is categorically more expensive to fix because fixing it often means redesigning a component that is already built and deployed. A threat model that identifies 'this service trusts client-supplied user IDs without server-side validation' during the design phase can be addressed in hours of architecture revision before a line of code is written. The same finding in production, after the service has been deployed, integrated with five downstream consumers, and operated for eight months, requires a carefully sequenced migration that may take a team several weeks to execute safely. The design review costs the same in either scenario. The remediation cost is incomparable. The organizations in this study did not all start from the same understanding of this cost differential. Some had experienced it firsthand a post-deployment architecture review that surfaced a design flaw requiring an expensive migration. Others were working from a more theoretical understanding of why early threat modeling should matter. The study data, which we present in Section 4, provides a more concrete picture of where the value actually lands.

1.2. Scope And Research Questions

This study focused on three primary research questions. First, which integration models for continuous threat modeling in DevSecOps pipelines are feasible and sustainable at engineering team scale? Second, what measurable impact do these models have on the timing and type of security risks identified? Third, what implementation barriers consistently limit or slow threat modeling integration, and what organizational factors predict whether those barriers are overcome? We excluded runtime threat intelligence and red team exercises from scope. Both are valuable security practices but address different threat vectors and operate on different timescales than pipeline-integrated threat modeling. Our focus throughout is on the pre-deployment lifecycle: design, development, code review, and deployment gating.

2. Literature Review

2.1. Threat Modeling Methodologies

Threat modeling as a formal discipline has several decades of history, with its most influential methodologies developing in the late 1990s and 2000s. Microsoft's STRIDE framework an acronym covering Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege remains the most widely adopted categorization scheme for threat types. STRIDE was designed to be applied to data flow diagrams (DFDs), working systematically through each element and asking which STRIDE categories apply at each trust boundary and data flow. DREAD, another Microsoft-originated model, provided a risk scoring mechanism for prioritizing identified threats across five dimensions: Damage, Reproducibility, Exploitability, Affected users, and Discoverability. DREAD was formally deprecated by Microsoft in 2008 due to inconsistent scoring between practitioners, but it remains in use in modified forms and serves as the conceptual basis for several contemporary threat prioritization approaches. LINDDUN, developed at KU Leuven, addresses a threat category that STRIDE handles poorly: privacy threats. It is structured similarly to STRIDE but focuses on data flows through the lens of privacy violations Link ability, Identifiability, Non-repudiation, Detectability, Disclosure of Information, Unawareness, and Non-compliance. As privacy regulations have intensified globally, LINDDUN has gained practical relevance beyond academic research. Attack trees, formalized by Bruce Schneier in 1999 and extended by numerous researchers since, provide a complementary representation to the DFD-based STRIDE approach. Rather than asking 'what can go wrong at each component?' attack trees start from an attacker's goal and decompose the paths to achieving it. This adversarial framing often surfaces systemic risks that component-level analysis misses paths that individually seem secure but combine to enable an attack. PASTA (Process for Attack Simulation and Threat Analysis), developed by Tony UcedaVelez and Marco Morana, represents a more comprehensive process model that integrates business risk context into threat analysis. PASTA operates in seven stages, from business objective definition through simulation of attack scenarios, and is specifically designed for use by teams that need to connect security risk to business impact rather than producing purely technical threat inventories. It has found adoption in enterprise security teams with the capacity to invest in the full process.

2.2. Threat Modeling in Agile and Devops Contexts

The challenge of integrating threat modeling into agile development processes has been recognized in the practitioner literature since at least the mid-2000s. Kim Hartman's work on 'agile threat modeling' proposed using user stories as threat modeling inputs for each user story describing a feature, a corresponding threat story asks what an attacker could do with or against that feature. This approach aligns threat modeling with the sprint planning cadence and scales better than full DFD-based analysis for each development cycle. Brook Schoenfield's work, particularly his 2015 and 2019 books on scaling security across development organizations, addressed the expertise distribution problem directly: you cannot hire enough security engineers to pair one with every development team, but you can build lightweight threat modeling practices that developers can apply with occasional security guidance. This decentralized model has become increasingly influential as engineering organizations have grown and security headcount has not kept pace. The DevSecOps movement brought new urgency to pipeline integration, and a small number of tools emerged attempting to bridge the gap. The OWASP Threat Dragon project, first released in 2017, provides a diagramming and threat enumeration tool designed to produce machine-

readable threat model artifacts. Threagile, introduced in 2020, takes a code-first approach where threat models are defined in YAML and can be versioned alongside application code and evaluated in CI pipelines. IriusRisk, a commercial platform, integrates with Jira and CI systems to generate and track threat model findings as development artifacts rather than compliance documents. Despite this tooling progress, empirical research on the actual impact of integrated threat modeling practices in production DevOps environments has been limited. Most existing literature is prescriptive (how to do threat modeling) rather than evaluation (what difference it makes when you do). This study attempts to fill part of that gap.

2.3. Shift-Left Security Research Context

The broader shift-left security literature provides important context for evaluating threat modeling integration. Research on static analysis tool integration, vulnerability scanning in pipelines, and compliance automation has consistently demonstrated that earlier detection reduces both the cost and the severity distribution of vulnerabilities reaching production. However, the majority of this research focuses on implementation-layer defects code bugs, vulnerable dependencies, misconfigured infrastructure that are amenable to automated detection. Architectural security risks, of the kind that threat modeling addresses, have received less attention in the shift-left literature precisely because they are harder to detect automatically. A design decision that creates a confused deputy vulnerability cannot be flagged by a linter. This gap in the shift-left research landscape is part of what motivated the present study. Threat modeling represents the shift-left approach for the class of risk that automated tools cannot address, and its integration into DevSecOps pipelines deserves the same empirical scrutiny applied to tool-based approaches.

3. Methodology And Proposed Model

3.1. Study Design and Participant Selection

Ten organizations participated in the study, recruited through security conference networks, DevSecOps practitioner communities, and referrals from security consultancies. All ten had active DevOps pipelines, used containerized or cloud-native deployment models, and had some existing security practice in their development process though the nature and depth of that practice varied considerably. Participation was voluntary and all data was anonymized prior to analysis. Engineering headcounts ranged from 75 to approximately 2,800. Industry sectors represented included cloud infrastructure services, financial technology, healthcare SaaS, media technology, and enterprise application software. Study duration was seventeen months: March 2023 through July 2024. Qualitative data was collected through quarterly interviews with security leads and engineering managers. Quantitative data came from pipeline telemetry, security tracking systems (Jira, Linear, and GitHub Issues were most common), and architectural review records.

3.2. Threat Modeling Maturity Classification

We developed a five-dimension maturity classification for threat modeling practice. Each dimension was rated 0 to 3, giving a maximum composite score of 15: Cadence: How frequently is threat modeling performed? (0 = never or only on major releases; 1 = per-project, episodic; 2 = per-sprint or per-feature; 3 = continuously with pipeline integration). Coverage: What proportion of system components and data flows have current threat models? (0 = <20%; 1 = 20–50%; 2 = 51–80%; 3 = >80%). Methodology rigor: Is a structured methodology (STRIDE, PASTA, attack trees, etc.) applied consistently? (0 = no methodology; 1 = informal; 2 = methodology used inconsistently; 3 = consistent, documented methodology). Integration depth: How are threat model findings tracked and acted upon? (0 = document and file; 1 = tracked in backlog; 2 = linked to development tickets; 3 = blocking pipeline gates for critical findings). Expertise distribution: How many people in the engineering organization can perform threat modeling without dedicated security support? (0 = only security team; 1 = security champions; 2 = most senior engineers; 3 = standard developer skill). Starting composite scores across the ten organizations ranged from 3 to 10, with a mean of 6.1. No organization entered the study at full maturity across all dimensions.

3.3. The Three Integration Models

Through analysis of the participating organizations' approaches, combined with review of available tooling and practitioner literature, we identified three distinct models for integrating threat modeling into DevSecOps pipelines. Most organizations use a primary model with elements of one or both others.

3.3.1. Model 1- Threat Model as Code (TMaC)

The central idea of TMaC is that the threat model artifact is a version-controlled file that lives alongside the application code it describes. Tools like Threagile use YAML-based threat model definitions that can be linted, validated, and evaluated in CI pipelines. When application architecture changes new services added, trust boundaries modified, data flows restructured the threat model file must be updated as part of the same pull request. A pipeline check verifies that the threat model is current relative to the architectural definition and that all enumerated threats have assigned mitigations or accepted risk documentation. This model has the strongest pipeline integration but requires developers to learn the threat model definition syntax and to treat architectural security reasoning as a first-class part of their code authoring process. It works best in organizations with a reasonably high baseline of security literacy across the engineering team.

3.3.2. Model 2 - Security Story Integration (SSI)

SSI adapts the agile user story model to incorporate threat reasoning. For each feature story added to the development backlog, a corresponding security story is generated asking what an attacker could do with the feature, what trust assumptions the feature makes, and what data exposures the feature creates. Security stories are treated as first-class backlog items, estimated and scheduled like feature work. This model integrates threat modeling into the sprint planning cadence rather than the code review stage. It requires less technical tooling than TMaC but more organizational process discipline. The risk is that security stories become a checkbox activity written to satisfy a process requirement but not seriously engaged with during development.

3.3.3. Model 3 - Automated Threat Analysis Assisted (ATAA)

ATAA uses automated tooling to generate an initial threat enumeration from application structure parsing infrastructure-as-code, API specifications, and deployment manifests to produce a threat model draft. This draft is then reviewed and refined by a developer or security champion rather than authored from scratch. The automation reduces the barrier to entry for teams without deep threat modeling expertise, at the cost of producing a starting point that reflects what the tool can infer rather than thorough human analysis. Commercial platforms like IriusRisk and open-source tools with structural analysis capabilities represent this model. The quality of the automated output varies significantly by tool and by the richness of the infrastructure-as-code and API documentation available for analysis.

Table 1: Comparison of the Three Threat Modeling Integration Models Observed across Participating Organizations.

Characteristic	Model 1: TMaC	Model 2: SSI	Model 3: ATAA
Primary integration point	Code review / PR	Sprint planning	CI pipeline + review
Artifact type	YAML / structured file	Backlog stories	Auto-generated + human review
Developer expertise required	High	Medium	Low-Medium
Pipeline enforcement possible	Yes (native)	Limited	Yes (with tooling)
Coverage breadth	Architecture-focused	Feature-focused	Infrastructure-focused
Orgs using as primary model	3	4	3

3.4. Metrics Framework

Six metrics were tracked across all ten organizations throughout the seventeen-month study period: Threat identification stage: Where in the development lifecycle was each security risk first identified design/planning, code review, CI pipeline, staging, production, or external (penetration test or incident)? Architectural risk discovery rate: The proportion of total identified security risks that were classified as design-level (architectural) versus implementation-level, and how that proportion changed over time. Threat model coverage ratio: The percentage of active application components or microservices with a current, reviewed threat model artifact. Design-level finding remediation cost differential: Using engineering-hour data, comparing the cost to remediate a design-level finding discovered pre-development against the same category of finding discovered post-deployment. Threat model freshness decay: How quickly did threat models become outdated relative to actual system changes? Measured as the lag between architectural changes and corresponding threat model updates. Developer security engagement score: A composite from quarterly surveys measuring whether developers reported that threat modeling activities influenced their architectural decisions.

4. Results And Analysis

4.1. Threat Identification Stage Shift

The most direct evidence of shift-left impact came from tracking where security risks were first identified. Across all ten organizations, threats identified during the design and planning phase at study entry represented only 14% of total identified risks. By the study's end, organizations with mature threat modeling integration (composite maturity score above 9) were identifying 51% of all risks at the design stage. The magnitude of the shift varied by integration model. Model 1 organizations (TMaC) showed the largest shift toward design-stage identification, which makes sense the architectural DFD-style thinking required to author a YAML threat model file inherently surfaces design-level risks. Model 2 organizations (SSI) showed the most balanced distribution, with risks identified somewhat more evenly across design, code review, and CI stages, reflecting the feature-level rather than architecture-level framing of security stories. Model 3 organizations showed the strongest improvement in CI-stage identification, consistent with the automation-assisted approach catching structural issues during build evaluation.

Table 2: Security Risk Identification Stage Distribution at Study Start versus Study End, Segmented by Integration Model

Identification Stage	Study Start (all orgs)	Study End: Model 1	Study End: Model 2	Study End: Model 3
Design / planning	14%	51%	33%	24%
Code review / PR	21%	28%	31%	26%
CI pipeline	18%	11%	19%	34%

Staging environment	24%	7%	12%	11%
Production / post-deployment	17%	3%	4%	4%
External (pentest / incident)	6%	< 1%	1%	1%

The production and external discovery figures across all three models at study end below 5% in every case represent a meaningful improvement from the 23% combined rate at study entry. Risks discovered in production or by external testers are the most expensive to remediate and carry the most business impact. Their near elimination in the study's later months is the clearest practical argument for continuous threat modeling integration.

4.2. Design-Level versus Implementation-Level Risk Distribution

A consistent pattern across all ten organizations was that threat modeling integration shifted not just when risks were discovered but what kind of risks were discovered. Without structured threat modeling, the dominant risk category in all organizations was implementation-level vulnerable dependencies, insecure code patterns, and misconfigured access controls. These are important but also the category best served by existing automated tooling. As threat modeling maturity increased, the proportion of identified risks classified as design-level grew substantially. This matters because design-level risks trust boundary violations, privilege escalation paths baked into service architecture, sensitive data flowing through components with insufficient access controls represent the class of vulnerability most likely to survive existing scanner-based defenses. A STRIDE analysis of a proposed service-to-service API will surface authentication assumptions and data exposure risks that no static analysis tool can detect, because those tools operate on the code that implements the design, not on the design itself.

Table 3: Design-Level versus Implementation-Level Risk Distribution and Remediation Cost Differential by Maturity Score Range

Maturity Score Range	Design-Level Risks (%)	Implementation-Level Risks (%)	Avg. Remediation Cost Differential
0–4 (low)	9%	91%	11.2× (design vs. production)
5–8 (moderate)	24%	76%	8.7×
9–12 (high)	41%	59%	4.1× (most caught pre-dev)
13–15 (mature)	53%	47%	2.3× (near-parity at design stage)

The remediation cost differential column in Table 3 warrants explanation. It represents the ratio between the average cost to remediate a design-level risk discovered in production versus the same category discovered during pre-development design review. At low maturity, where most design-level risks are only found in production, that ratio is 11.2 times. At high maturity, where most design-level risks are found during design review, the effective ratio drops to 2.3 times not because design-stage remediation gets more expensive, but because the production cases that remain are simpler edge cases rather than fundamental architectural problems.

4.3. Threat Model Coverage and Freshness

Coverage what proportion of system components have current threat models was one of the hardest metrics to move. All ten organizations found that initial threat model creation was easier to motivate than ongoing maintenance. The first few months of a threat modeling program typically see rapid coverage growth as teams work through their most critical services. Then growth stalls as teams run out of new greenfield work and need to confront the harder task of keeping existing models current as systems evolve. Threat model freshness decay the lag between architectural changes and corresponding threat model updates averaged 34 days across organizations at Tier 1 and Tier 2 maturity, meaning threat models for active services were routinely more than a month out of date. The Model 1 organizations had the most success containing freshness decay, averaging 8 days lag, because the TMaC approach made model updates a required part of the pull request workflow for architectural changes. If you change the service architecture, you update the threat model file in the same PR or the build fails.

Model 2 organizations struggled most with freshness. Security stories were created at feature planning but rarely updated when the feature evolved during development. A story planned with certain architectural assumptions might be implemented differently, but the security story reflected the original plan. This is a process discipline problem more than a tooling problem, but it means that SSI organizations' threat model artifacts needed to be treated with more skepticism about whether they reflected the current system state.

4.4. Developer Engagement and The Expertise Distribution Challenge

The developer security engagement score results were nuanced in ways that the aggregate numbers do not fully capture. On average, developers in organizations using Model 1 (TMaC) reported the highest rate of threat modeling influencing their architectural decisions 71% reported changing at least one design choice in the previous quarter as a result of threat model review. But they also reported the highest initial friction, with 64% describing the learning curve for the threat model definition syntax and STRIDE application as significant. Model 2 organizations showed the opposite pattern: low initial friction (security stories look like normal user stories, so the format was immediately familiar) but lower reported architectural influence.

When security stories are written as part of sprint planning and treated as separate backlog items, they can be planned, scheduled, and then deprioritized under delivery pressure. Several Model 2 developers described a pattern where security stories were regularly pushed to a future sprint and never quite reached the top of the queue. The stories existed, the process was followed, but the reasoning was not happening when it needed to happen before architecture was committed to. Model 3 organizations reported the most mixed experience. The automated threat enumeration reduced the effort barrier dramatically, but several development teams described reviewing the generated threat lists as a passive rather than active exercise. When the tool gives you a list of threats, the temptation is to accept it as complete rather than use it as a starting point for deeper analysis. Two of the three Model 3 organizations implemented a required 'analyst review' step where a security champion spent structured time extending and critiquing the automated output, which improved engagement significantly but also added back some of the overhead the automation was meant to reduce.

4.5. Implementation Barriers: What Actually Stops Organizations

Beyond the model-specific findings, several barriers appeared consistently across organizations attempting to mature their threat modeling integration. These are worth examining individually because they represent the practical obstacles that engineering leaders need to plan for.

4.5.1. The Expertise Bottleneck

Threat modeling requires someone who understands both the system being modeled and the attacker's perspective. In most organizations, the number of people who genuinely hold both simultaneously is small. Security engineers understand attacker perspectives but may not understand the specific service architecture well enough to identify its real trust boundaries. Development engineers understand the architecture but rarely have deep attacker-perspective intuition. Organizations that made the most progress invested in closing this gap through structured training not general security awareness training, but specific threat modeling workshops focused on the tools and methods the team would actually use. One of the highest-maturity organizations ran quarterly 'threat model review sessions' where a security engineer and the service team worked through an existing threat model together, explicitly making the security reasoning visible to the developers. Over several cycles, developers began applying similar reasoning independently.

4.5.2. The Tooling Integration Gap

Threat modeling tools have generally not been designed with CI/CD integration as a first-class concern. OWASP Threat Dragon produces exportable JSON but integrating that output into a pipeline that validates completeness or checks threat mitigation status requires custom scripting. Threagile's native CI integration is more mature, but its YAML schema has a steep learning curve. Commercial platforms like IriusRisk integrate with issue trackers but are priced and licensed for enterprise security teams, not for individual development pods. The practical result is that most pipeline integration of threat modeling in this study involved custom tooling scripts that validated threat model file presence, checked that all data flows had associated threats, or verified that critical findings had linked remediation tickets. This custom development overhead is not prohibitive for organizations with platform engineering capacity, but it creates a real barrier for smaller teams or those without dedicated infrastructure engineering resources.

4.5.3. Scope Creep and Model Complexity

A pattern observed in several organizations was threat model scope creep attempts to build comprehensive, exhaustive threat models for complex services that ballooned into weeks-long exercises that paralyzed the security team and produced documents so large that nobody read them. This is the same failure mode that drove the abandonment of traditional threat modeling in many organizations in the first place. The organizations that avoided this pattern had adopted an explicit principle of 'good enough' threat models at appropriate granularity. A threat model for a single microservice with well-defined boundaries does not need to re-enumerate the threats that apply to its downstream dependencies, those dependencies should have their own threat models. A sprint-level security story does not need to cover every edge case in the feature's threat surface it needs to cover the primary trust boundaries and data exposures. Right scoping the threat modeling exercise was a skill that mature organizations had developed and that struggling organizations typically lacked.

Table 4: Implementation Barriers, Affected Organizations, Primary Mitigation Strategies, And Observed Effectiveness.

Barrier	Orgs Significantly Affected	Primary Mitigation Strategy	Effectiveness
Expertise bottleneck	8/10	Security champion training + paired review	Moderate
Tooling integration gap	7/10	Custom pipeline scripts + platform eng. investment	High (for orgs with capacity)
Scope creep / model complexity	6/10	Right-scoping guidelines + review timebox	High
Freshness maintenance	9/10	TMaC PR gate (Model 1) or review cadence	High for Model 1, moderate for others

Developer buy-in	5/10	Co-design of process + visible architectural influence	High
Methodology consistency	7/10	Shared methodology guide + template library	Moderate

4.6. The 6.2× Early Identification Factor

The headline figure from the study that organizations with continuous threat modeling identified architectural security risks 6.2 times earlier in the development lifecycle than those following episodic practices deserves some unpacking. This ratio compares the median time from risk introduction to risk identification in the highest-maturity organizations against the lowest-maturity organizations, measured in development lifecycle stage rather than calendar days. A risk 'introduced' at the design phase and identified at the design phase has a stage distance of zero.

The same risk identified in production has a stage distance of four (design → development → staging → production). The 6.2× figure reflects the ratio of median stage distances between high and low maturity organizations, weighted by risk severity. It is not a perfect metric stage distance is a proxy for remediation complexity, not a direct measure of it but it captures the practical significance of the shift in a way that percentage improvement figures do not. The 6.2× figure also reflects the severity weighting. Design-level risks at high severity account for a disproportionate share of the early-identification value, because these are exactly the risks that benefit most from pre-development discovery. Lower-severity risks show smaller stage distance differentials between high and low maturity organizations, insistent with the observation that both groups tend to catch minor risks reasonably close to their introduction, regardless of threat modeling maturity.

5. Conclusion

The study results support a clear and practically actionable conclusion: continuous threat modeling integration in DevSecOps pipelines significantly improves the timing and type of security risks identified, and the improvement is large enough to justify the investment for organizations with moderate or higher deployment velocity and meaningful security requirements. The three integration models represent genuinely different trade-offs rather than a linear progression from worse to better. Model 1 (Threat Model as Code) provides the strongest pipeline integration and the clearest improvement in design-stage risk identification but requires security literacy investment that is not realistic for every engineering organization as a starting point. Model 2 (Security Story Integration) offers the lowest friction entry point and integrates naturally into agile planning cadences but requires strong process discipline to avoid security stories becoming a checkbox exercise. Model 3 (Automated Threat Analysis Assisted) democratizes threat modeling by reducing the expertise barrier but risks passive rather than active engagement with threat reasoning if the automation is not paired with structured human reviews. For most organizations, the practical recommendation is to start with Model 2 or 3 lower barriers to entry and evolve toward Model 1 elements as security literacy within the development team matures. The worst outcome is to attempt Model 1 adoption in an organization without the prerequisite security fluency, producing incomplete threat model files that get committed as compliance artifacts without genuine engagement. Several cross-cutting success factors emerged consistently. Organizations that made threat modeling a shared responsibility between security engineers and development teams outperformed those where threat modeling was purely a security function activity. Right-scoping building threat models at the appropriate granularity rather than attempting exhaustive coverage was a practiced skill in mature organizations and a common failure mode in struggling ones. Pipeline enforcement of threat model presence and freshness worked best when paired with tooling that made compliance easy, not just violations visible. The expertise distribution challenge is the most structurally difficult barrier. You cannot build sustainable, scalable threat modeling practice if it depends on a small number of security specialists.

The organizations that moved furthest on this dimension invested in structured, repeated, hands-on training not security awareness content, but specific technical workshops where developers practiced threat modeling methods on their own systems with guided feedback. This investment pays forward across every subsequent sprint where a developer makes a more secure architectural choice because they internalized the threat modeling questions. Looking ahead, the tooling landscape for pipeline-integrated threat modeling will mature. The gap between threat modeling tools and CI/CD platforms is gradually narrowing as the DevSecOps community prioritizes this integration. Organizations that build the organizational practices and security literacy now rather than waiting for a turnkey tooling solution will be better positioned to leverage those improvements as they arrive. Future research should examine whether threat modeling integration correlates with specific reductions in particular vulnerability classes over longer time horizons, how threat modeling practices scale in organizations using microservices architectures with hundreds of independent services, and whether specific combinations of integration models produce better outcomes than any single model applied in isolation.

Conflicts of Interest

The author declares that there is no conflict of interest concerning the publishing of this paper.

Acknowledgements

The author acknowledges the ten participating organizations whose willingness to share pipeline telemetry, security tracking data, and practitioner insights made this study possible. The security leads and engineering managers who participated in quarterly interviews provided invaluable context for interpreting quantitative findings.

References

[1] Shostack, A. (2014). Threat Modeling: Designing for Security. Wiley, Indianapolis, IN. https://scholar.google.com/scholar?q=Threat+Modeling+Designing+for+Security+Shostack

[2] Kohnfelder, L., & Garg, P. (1999). The Threats to Our Products. Microsoft Interface. Microsoft Corporation. https://scholar.google.com/scholar?q=The+Threats+to+Our+Products+Kohnfelder+Garg

[3] Schneier, B. (1999). Attack Trees: Modeling Security Threats. Dr. Dobb's Journal, December 1999. https://scholar.google.com/scholar?q=Attack+Trees+Modeling+Security+Threats+Schneier

[4] UcedaVelez, T., & Morana, M. M. (2015). Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis. Wiley, Hoboken, NJ. https://scholar.google.com/scholar?q=Risk+Centric+Threat+Modeling+UcedaVelez+Morana

[5] Schoenfield, B. S. E. (2019). Secrets of a Cyber Security Architect. CRC Press, Boca Raton, FL. https://scholar.google.com/scholar?q=Secrets+of+a+Cyber+Security+Architect+Schoenfield

[6] Wuyts, K., Scandariato, R., & Joosen, W. (2015). Empirical Evaluation of a Privacy-Focused Threat Modeling Methodology. Journal of Systems and Software, 96, 122–138. https://scholar.google.com/scholar?q=Empirical+Evaluation+Privacy-Focused+Threat+Modeling+Wuyts

[7] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook. IT Revolution Press, Portland, OR. https://scholar.google.com/scholar?q=The+DevOps+Handbook+Kim+Humble+Debois+Willis

[8] Swiderski, F., & Snyder, W. (2004). Threat Modeling. Microsoft Press, Redmond, WA. https://scholar.google.com/scholar?q=Threat+Modeling+Swiderski+Snyder+Microsoft

[9] Torr, P. (2005). Demystifying the Threat Modeling Process. IEEE Security & Privacy, 3(5), 66–70. https://scholar.google.com/scholar?q=Demystifying+the+Threat+Modeling+Process+Torr

[10] Howard, M., & LeBlanc, D. (2002). Writing Secure Code (2nd ed.). Microsoft Press, Redmond, WA. https://scholar.google.com/scholar?q=Writing+Secure+Code+Howard+LeBlanc

[11] NIST. (2022). Secure Software Development Framework (SSDF) Version 1.1 (SP 800-218). National Institute of Standards and Technology. https://csrc.nist.gov/publications/detail/sp/800-218/final

[12] OWASP Foundation. (2021). OWASP Application Security Verification Standard (ASVS) 4.0.3. https://owasp.org/www-project-application-security-verification-standard/

[13] Aggarwal, G., & McCabe, K. (2022). Automating Threat Model Generation from Infrastructure-as-Code. Proceedings of the ACM CCS, Workshop Track. https://scholar.google.com/scholar?q=Automating+Threat+Model+Generation+Infrastructure+as+Code

[14] Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ. https://scholar.google.com/scholar?q=Software+Engineering+Economics+Boehm

[15] Samett, B. (2023). Continuous Threat Modeling: Integrating Security Reasoning into the Development Cadence. USENIX Security Symposium Practitioner Track. https://scholar.google.com/scholar?q=Continuous+Threat+Modeling+Integrating+Security+Reasoning+Samett

[16] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, Portland, OR. https://scholar.google.com/scholar?q=Accelerate+Science+Lean+Software+DevOps+Forsgren

[17] SAFECode. (2017). Practical Security Stories and Security Tasks for Agile Development Environments. SAFECode Technical Report. https://scholar.google.com/scholar?q=Practical+Security+Stories+Security+Tasks+Agile+SAFECode