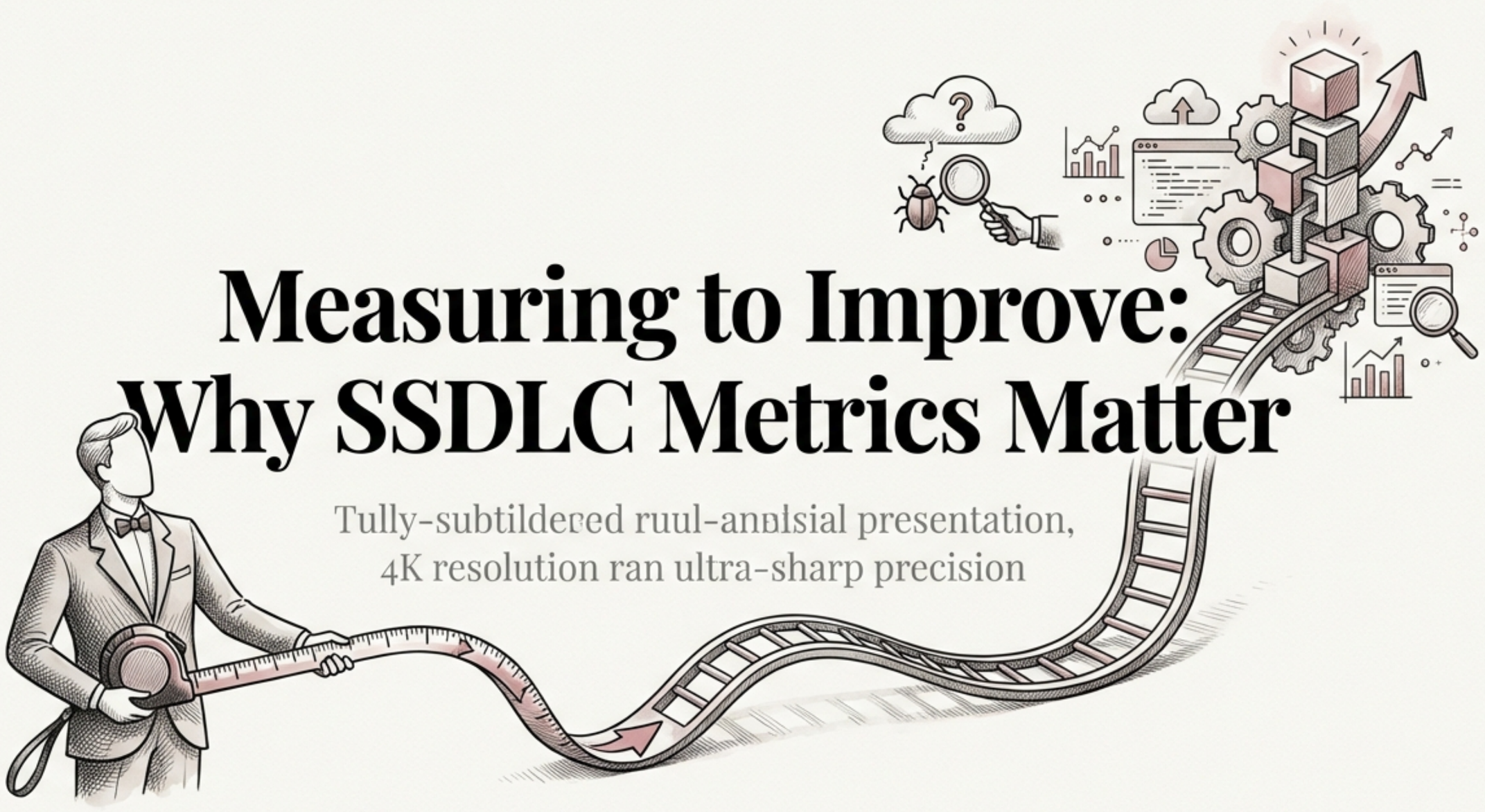
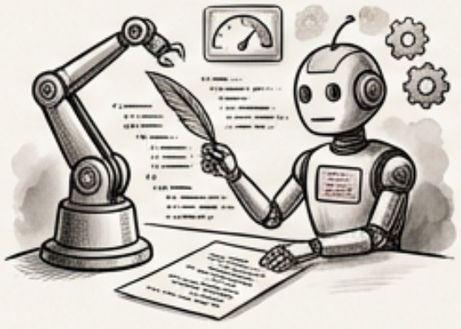
Measuring to Improve: Why SSDLC Metrics Matter

Tully-subtildered ruul-anɒlsial presentation,
4K resolution ran ultra-sharp precision



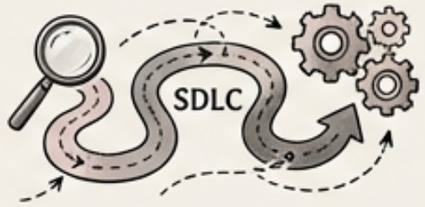
Measuring to Improve: Why SSDLC Metrics Matter



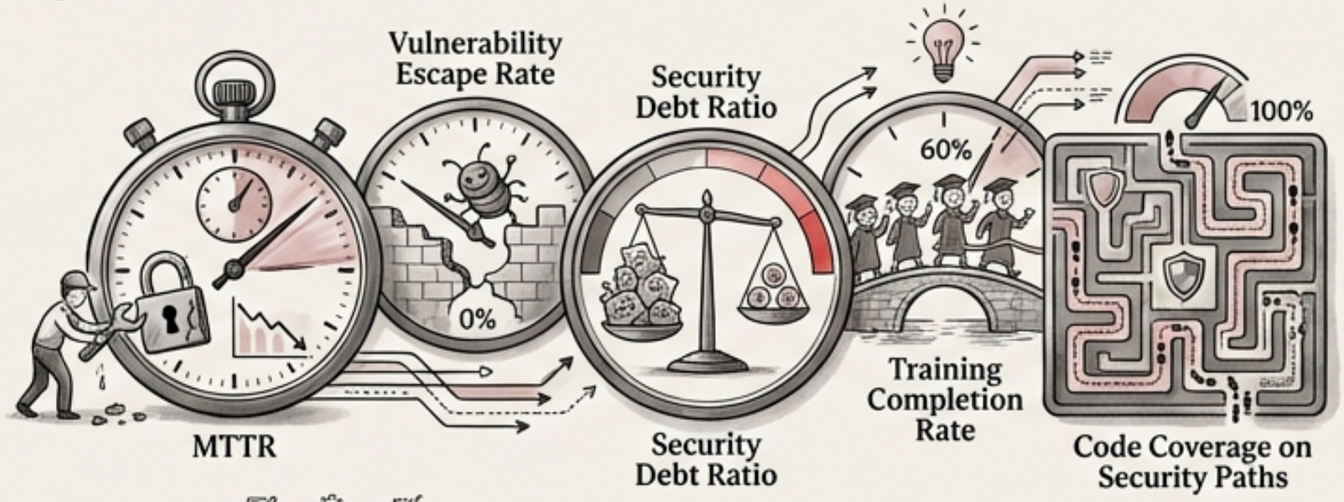
- An SSDLC program without metrics operates blindly, hindering effective security improvements.
- "What gets measured gets improved" highlights the fundamental principle of data-driven security.
- For AI-augmented teams, new metrics are essential to track AI code quality and tool effectiveness.
- These metrics also help assess the security impact of AI adoption within the development lifecycle.
- By measuring and monitoring these metrics, we can continuously refine our SSDLC program and enhance its effectiveness.

ESSENTIAL SSDLC METRICS: YOUR SECURITY DASHBOARD

- A core set of metrics is crucial to tracking and improving your SDLC.



- These essential metrics are Mean Time to Remediate (MTTR), Vulnerability Escape Rate, Security Debt Ratio, Training Completion Rate, and Code Coverage on Security Paths.

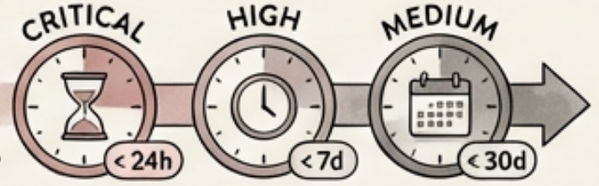


- These measurements help visualize security risk across the whole SDLC.

MTTR: Speeding Up Vulnerability Remediation

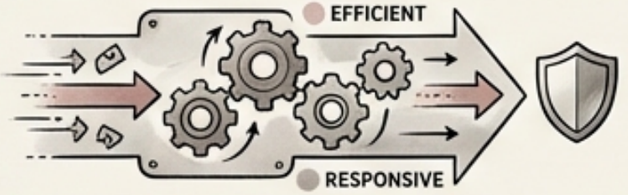
- Mean Time to Remediate (MTTR) measures the average time from vulnerability discovery to verified fix.

- Target MTTR: Critical vulnerabilities < 24 hours, High vulnerabilities < 7 days, Medium vulnerabilities < 30 days.

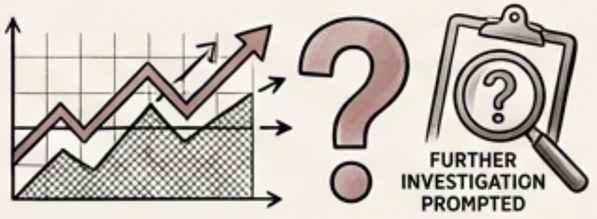


- Faster MTTR reduces the window of opportunity for attackers to exploit vulnerabilities.

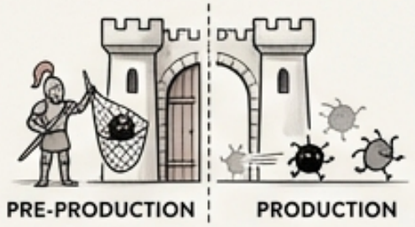
- Shorter MTTR generally indicates a more efficient and responsive security process.



- Consistently exceeding MTTR targets for each severity level should prompt further investigation.



Vulnerability Escape Rate: Preventing Production Leaks



Vulnerability Escape Rate measures the percentage of vulnerabilities that reach production environments versus those caught pre-production.

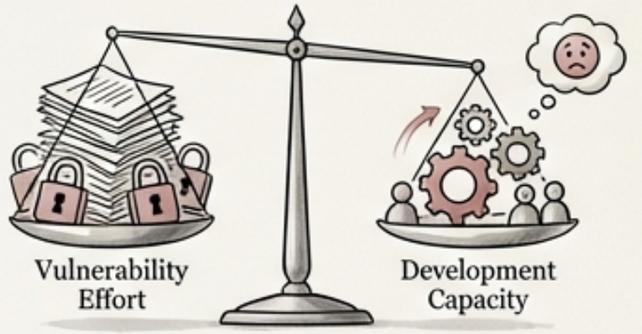
Target Vulnerability Escape Rate: Less than 5%.

A high escape rate suggests weaknesses in pre-production security controls.

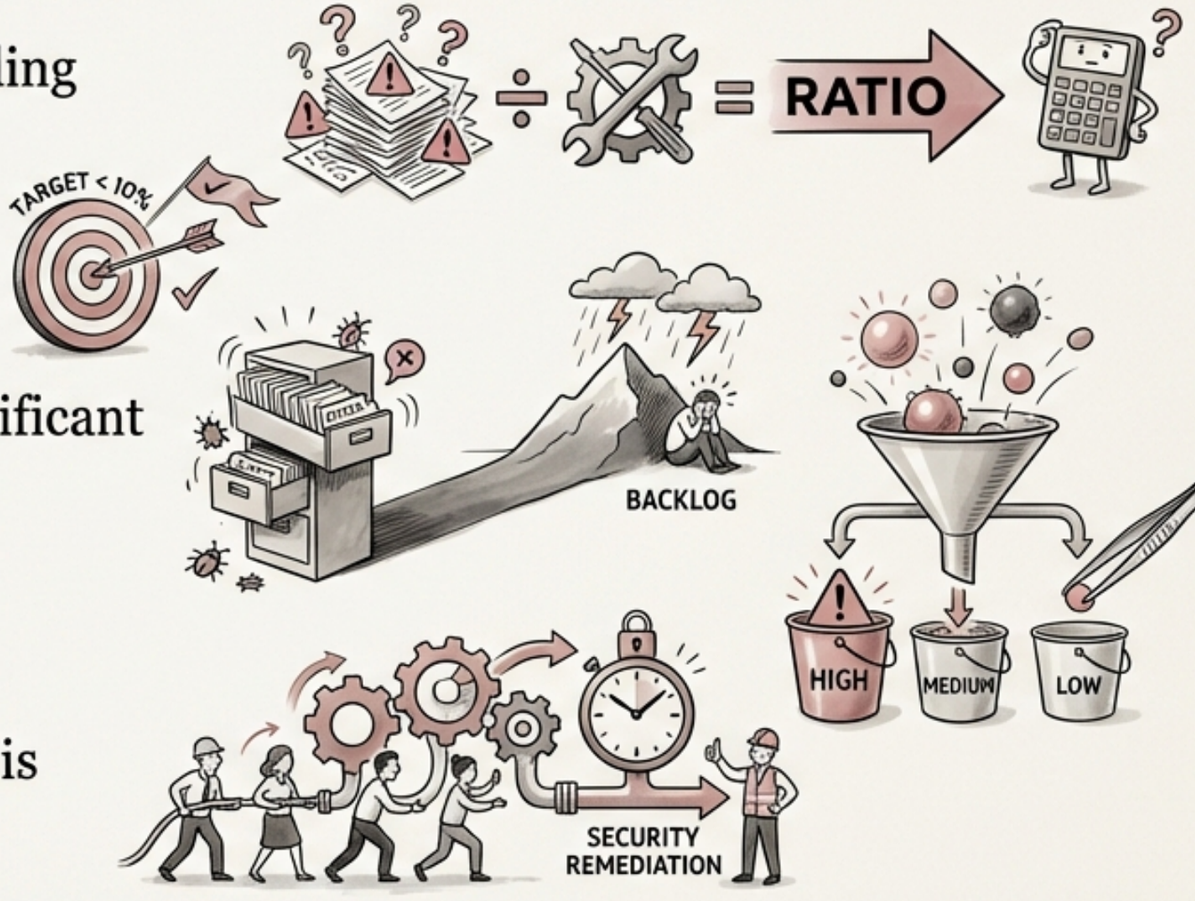
Focus on improving static analysis, dynamic analysis, and penetration testing efforts.

Escaped vulnerabilities represent a significantly higher risk due to their exposure in live environments.

Security Debt Ratio: Managing Remediation Backlog



- **Security Debt Ratio** represents the outstanding vulnerability remediation effort versus total development capacity.
- **Target Security Debt Ratio**: Less than 10%.
- A high security debt ratio can indicate a significant backlog of unresolved vulnerabilities.
- Prioritize remediation efforts based on vulnerability severity and potential impact.
- Ensure that sufficient development capacity is allocated to security remediation activities.

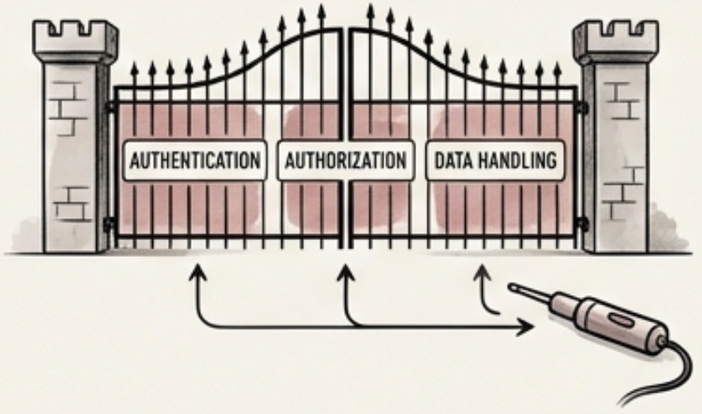


Training Completion Rate: Investing in Developer Security Knowledge

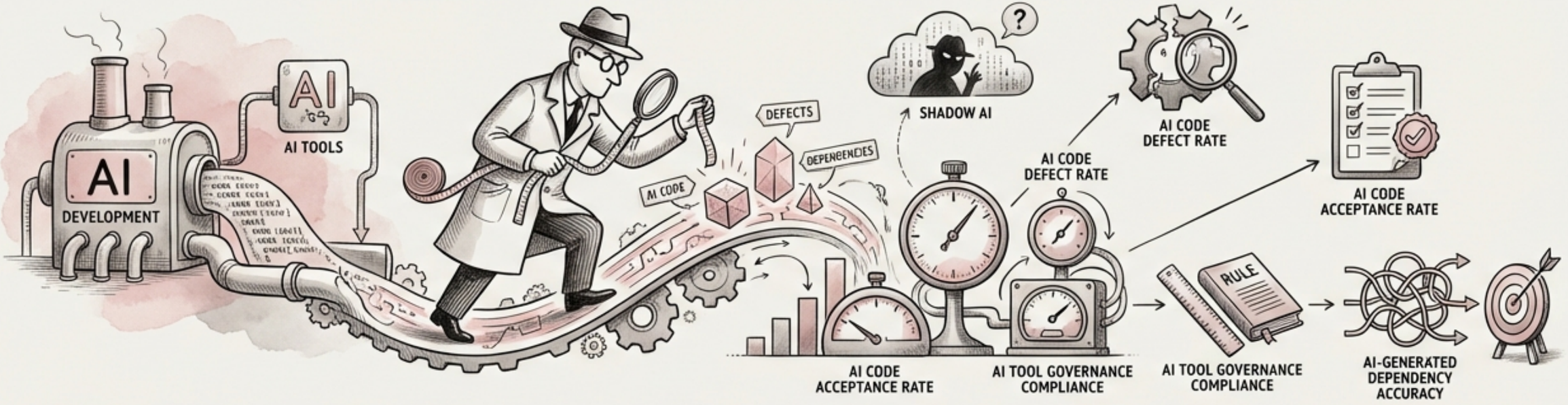
- Training Completion Rate measures the percentage of developers completing required security training.
- Target Training Completion Rate: 100%.
- Well-trained developers are better equipped to identify and prevent security vulnerabilities.
- Regular security training programs should cover secure coding practices, common vulnerabilities, and security tools.
- Track individual training completion rates and follow up with developers who are behind schedule.

Code Coverage on Security Paths: Ensuring Thorough Testing

- **Code Coverage on Security Paths** measures test coverage specifically for authentication, authorization, and data handling code.
- **Target Code Coverage on Security Paths:** Greater than 90%.
- Adequate test coverage helps identify potential vulnerabilities and ensures code behaves as expected.
- Focus on testing critical security functionalities such as login processes, access control mechanisms, and data validation routines.
- Use code coverage tools to identify areas with insufficient test coverage.

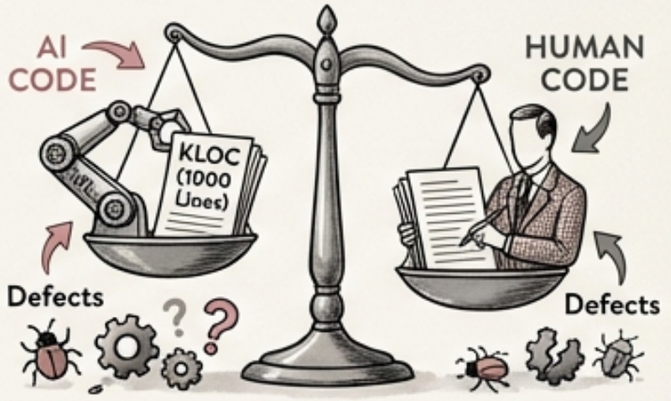


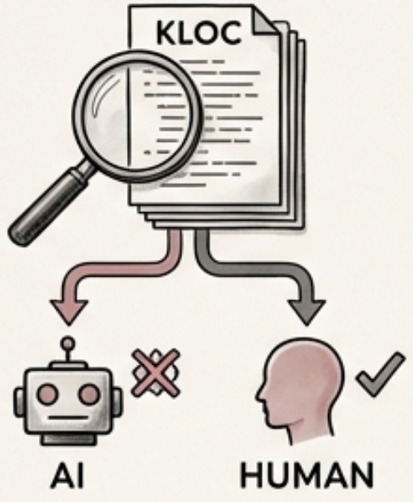
AI-Specific Metrics: Adapting to the New Landscape



- The increasing use of AI in development necessitates new metrics for monitoring AI code quality and governance.
- AI Code Defect Rate, AI Code Acceptance Rate, AI Tool Governance Compliance, Shadow AI Detection Rate, and AI-Generated Dependency Accuracy provide insight on these areas.
- These metrics help ensure the security and reliability of AI-generated code and tools.
- A comprehensive measurement approach is crucial for managing the risks associated with AI adoption.
- These measurements are important to track since more companies are embracing AI for development.

AI Code Defect Rate: Comparing AI vs. Human Code Quality

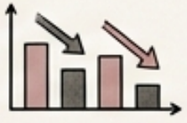




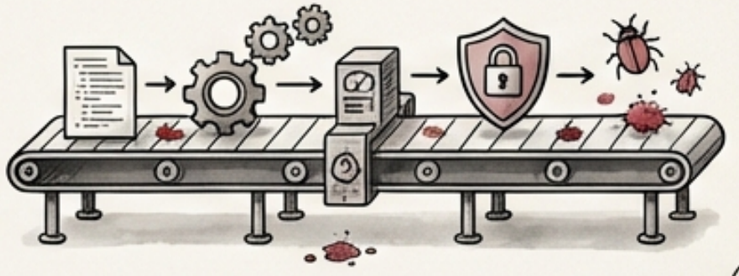
- **AI Code Defect Rate** measures vulnerabilities per KLOC (thousand lines of code) in AI-generated code versus human-written code.
- Tracking these rates separately helps identify quality trends and potential risks associated with AI code.

- A significantly higher defect rate in AI-generated code may indicate the need for more rigorous review processes.

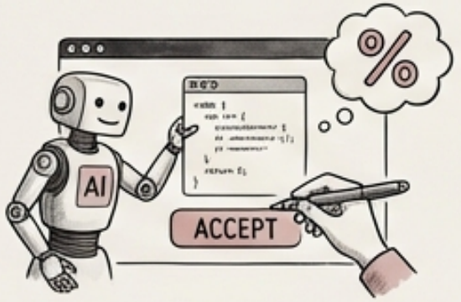
- Compare defect rates across different AI models or tools to identify best practices.

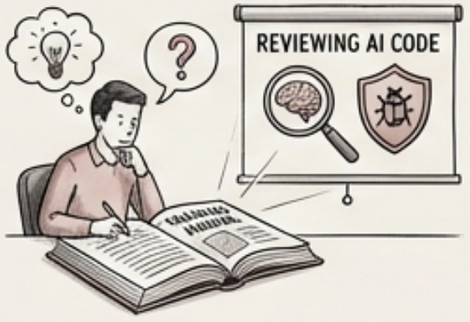


- Consider integrating automated security scanning tools into the AI code generation pipeline.

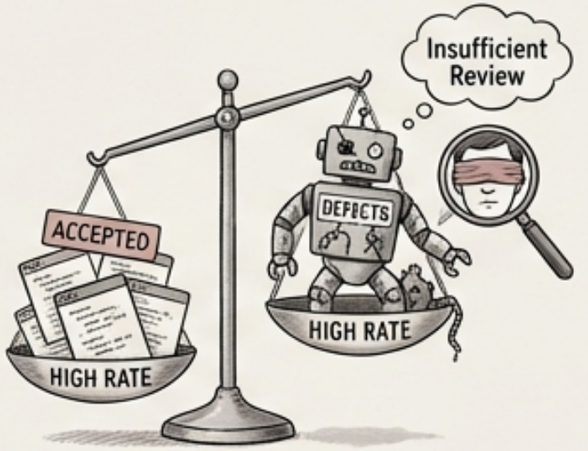


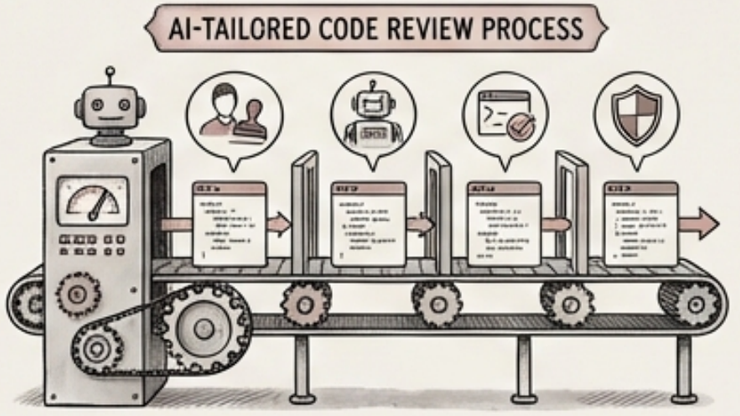
AI Code Acceptance Rate: Spotting Potential Blind Spots



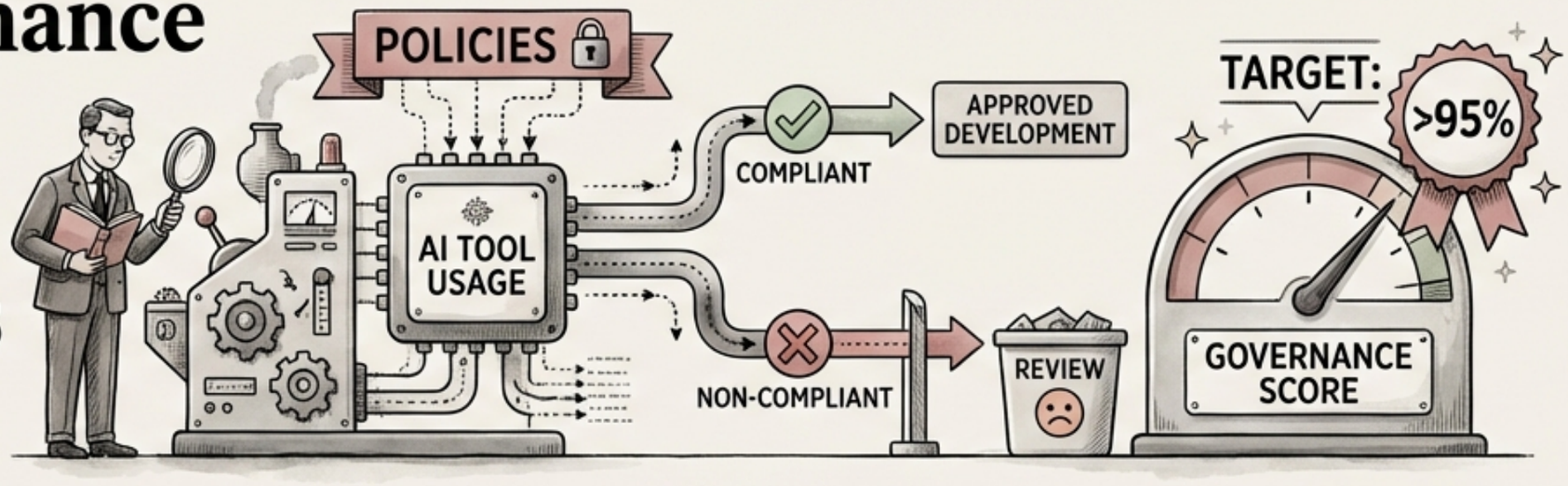


- **AI Code Acceptance Rate** measures the percentage of AI suggestions accepted by developers.
- A high acceptance rate coupled with a high defect rate suggests **insufficient review** of AI-generated code.
- Encourage developers to **carefully review** and **validate** all AI suggestions before accepting them.
- Provide **training** on how to effectively review AI-generated code and identify potential vulnerabilities.
- Implement code review processes specifically **tailored** to AI-generated code.





AI Tool Governance Compliance: Enforcing AI Usage Policies



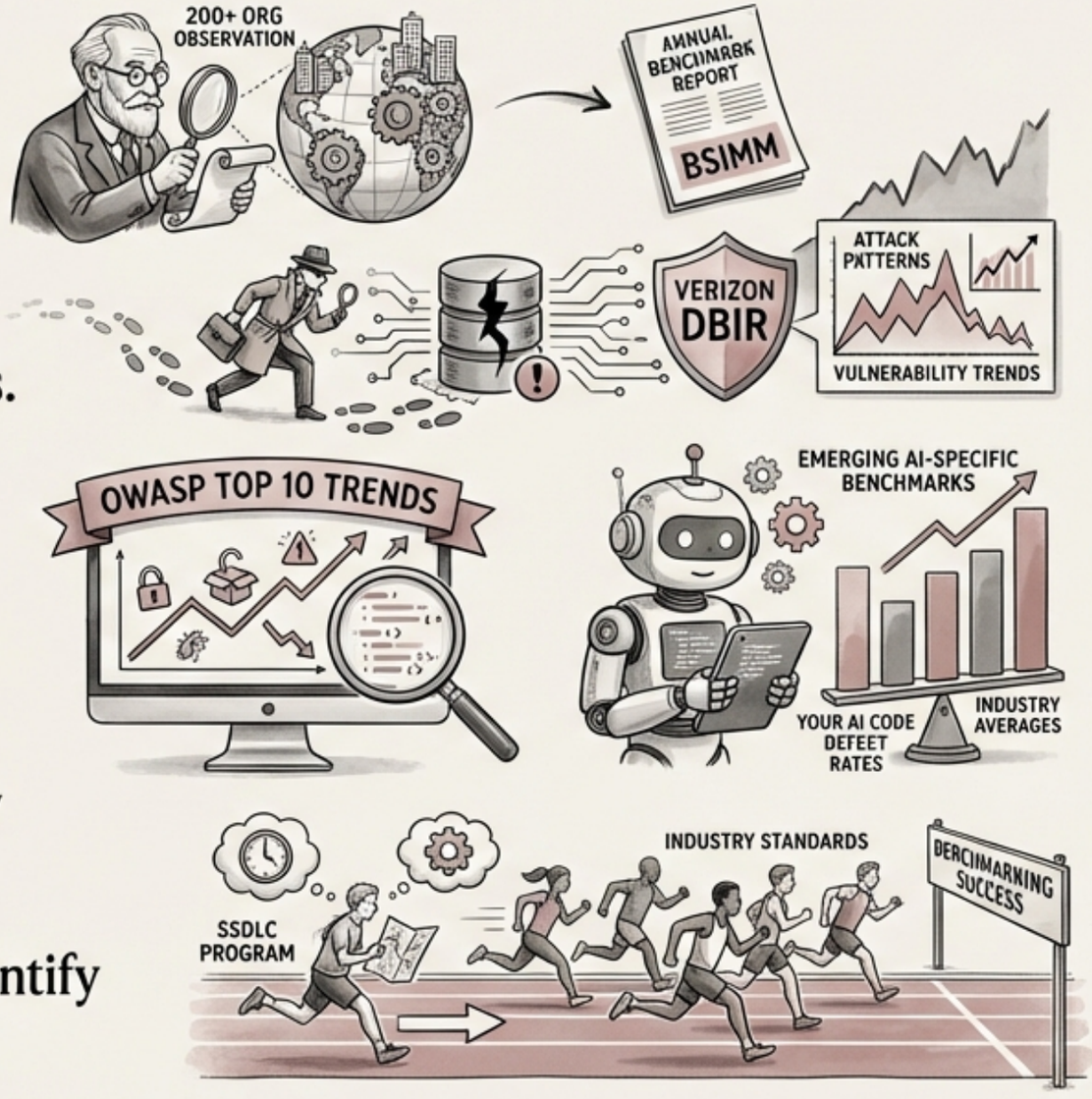
- AI Tool Governance Compliance measures the percentage of AI tool usage that follows approved policies.
- Target AI Tool Governance Compliance: Greater than 95%.
- Establish clear policies regarding the use of AI tools in the development process.
- Monitor AI tool usage to ensure compliance with these policies.
- Enforce policies related to data privacy, security, and ethical considerations.

CONTINUOUS IMPROVEMENT WITH OWASP SAMM

- OWASP Software Assurance Maturity Model (SAMM) provides a framework for assessing and improving your SSDLC.
- SAMM includes 5 business functions, 15 practices, and 3 maturity levels for each practice.
- Conduct annual full assessments and quarterly targeted assessments to identify gaps and prioritize improvements.
- Use SAMM to create a multi-year SSDLC improvement plan with measurable milestones.
- SAMM provides a clear roadmap for enhancing your organization's software assurance capabilities.

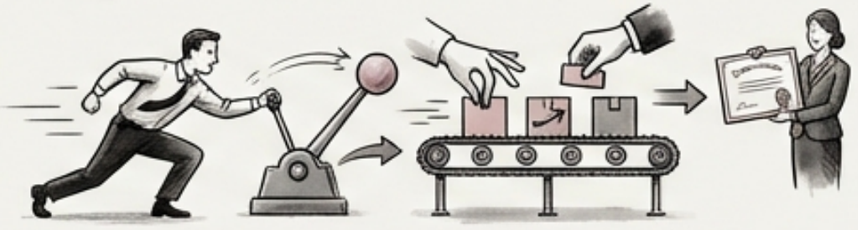
Benchmarking: Learning from Industry Trends

- BSIMM provides a descriptive model based on observation of 200+ organizations, offering industry benchmarks in its annual report.
- Verizon DBIR offers annual breach data showcasing attack patterns and vulnerability trends.
- OWASP Top 10 trends allows you to track which vulnerability categories are increasing or decreasing in your codebase.
- Emerging AI-specific benchmarks allow you to compare your AI code defect rates against industry averages as data becomes available.
- Benchmarking against industry standards helps identify areas where your SSDLC program may be lagging.



THE CONTINUOUS IMPROVEMENT CYCLE: PLAN, DO, CHECK, ACT

- The continuous improvement cycle involves four key steps: Plan, Do, Check, and Act.
- Plan: Identify improvement areas based on metrics, assessments, and incident reviews.
- Do: Implement improvements, such as tool changes, process updates, and training programs.
- Check: Measure the impact of improvements against baseline metrics.
- Act: Standardize successful improvements and abandon ineffective ones.



Actionable Metrics for Developers: Driving Security Ownership

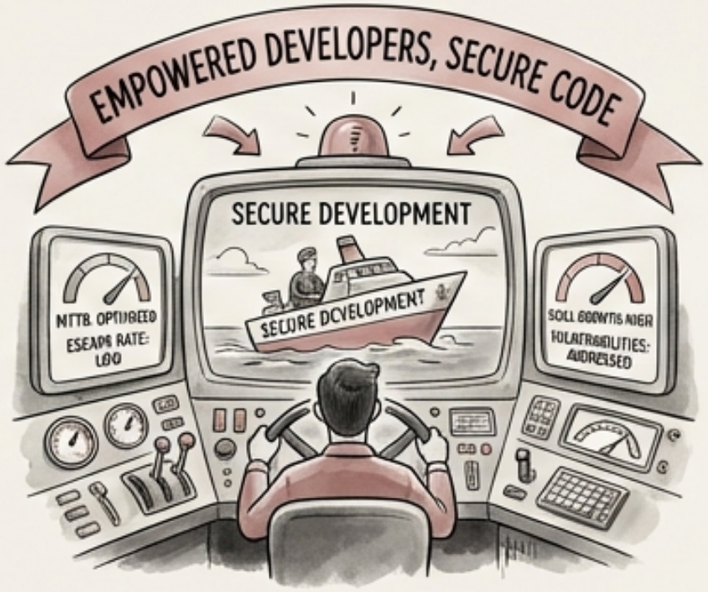
- Development teams benefit from personalized metrics, team comparisons, and improvement suggestions.

- Examples of helpful data include MTTR trends, escape rates, tool effectiveness, and team workload.

- Provide actionable training recommendations based on individual performance and skill gaps.

- Foster a culture of continuous learning and security ownership among developers.

- Use metrics to empower developers to proactively identify and address security vulnerabilities.



Thank You

- Questions?