



MANICODE

SECURE CODING EDUCATION

Artificial Intelligence Secure Code Creation



A Little Background Dirt...

jim@manicode.com

 @manicode

- Former OWASP Global Board Member
- 25+ years of software development experience
- Author of Iron-Clad Java, Building Secure Web Applications from McGraw-Hill/Oracle-Press
- OWASP Project Leader
 - OWASP Cheat Sheet Series
 - OWASP Application Security Verification Standard

AI Code Generation Learning Objectives

The rise of AI in coding

Benefits and challenges of AI-generated code

Security considerations in AI code generation

Best practices for prompt engineering and validation

AI Secure Code Generation Introduction

Artificial Intelligence in Code Creation

- AI's growing role in coding
- Importance of secure AI-generated code
- Setting the stage: security in AI code generation



Benefits of AI Code Generation

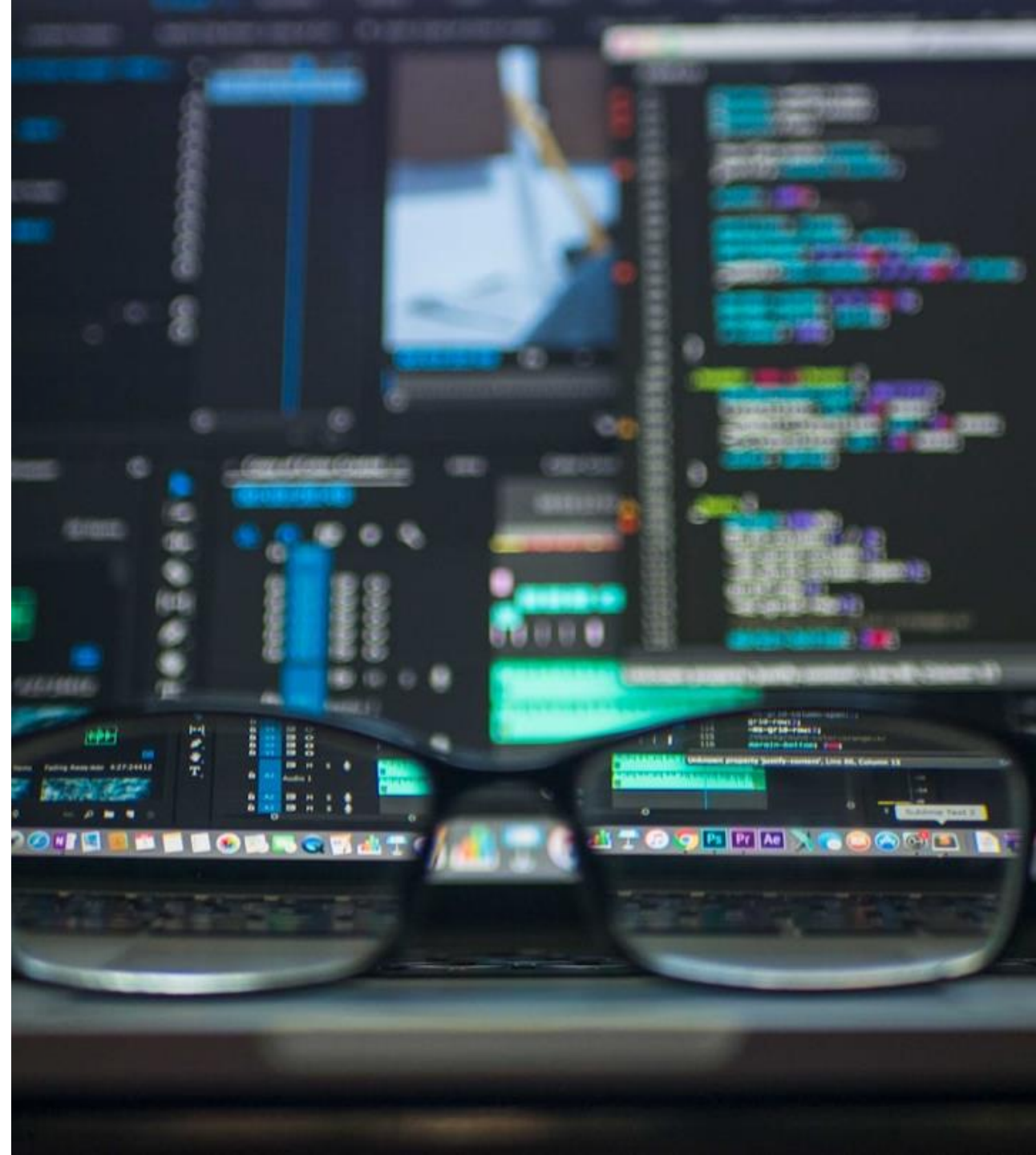
Increased productivity

Code suggestions

Learning aid

Reduced errors

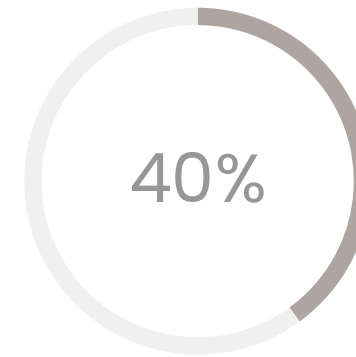
Documentation assistance



Increased Development Speed & Error Reduction

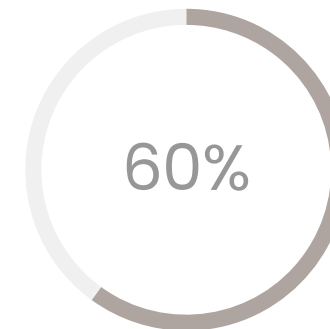
- **Development Speed**

- AI-powered tools automate tasks and offer smart suggestions, increasing coding speed by 60%.



- **Consistency and Reducing Errors**

- Following standards and best practices with AI tools improves code consistency and reduces errors by 40%.



Most Important Benefit Efficiently Moving In and Out of Code

- Enhanced Code Memory
 - Seamless flow reducing time to navigate back to the original thought process
- “How was your weekend?”
 - Quickly rebuild thought processes and continue coding efficiently even after interruptions like meetings or breaks

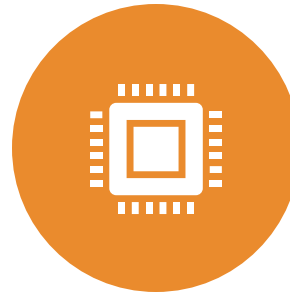


Challenges in AI Secure Code Generation

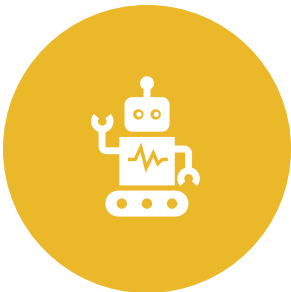
Challenges in AI Generated Code



**AI Models Often
Generate Insecure Code**



**AI-Generated Code
Contains Security
Vulnerabilities**



**Lack of Built-In Security
Awareness in AI**



**Over-Reliance on Raw AI
Prompts**

Common Misuse Patterns

- **Resource Leak:** Failure to close streams or resources
- **Insecure Data Handling:** Storing sensitive info in mutable objects
- **Exception Handling:** Generic catch blocks
- **Concurrency:** Non-atomic compound actions
- **API Contract Violations:** Ignoring return values, parameter constraints
- **Misconfigured Security Contexts:** Incorrect use of crypto APIs
- **Dependency Hell:** Conflicting library versions



Why Should We Care?

- Escalating Complexity
- Dependence on Third-party Libraries
- Prevalence of Code Generation Tools
- Risks of API Misuse
- CWEs and OWASP Top 10

AI Secure Code Generation Lifecycle

Secure AI Code Lifecycle



CREATE CLEAR SECURITY
RULES



PROMPT ENGINEERING
FOR SECURITY



SECURE AI CODE
GENERATION AND
VERIFICATION

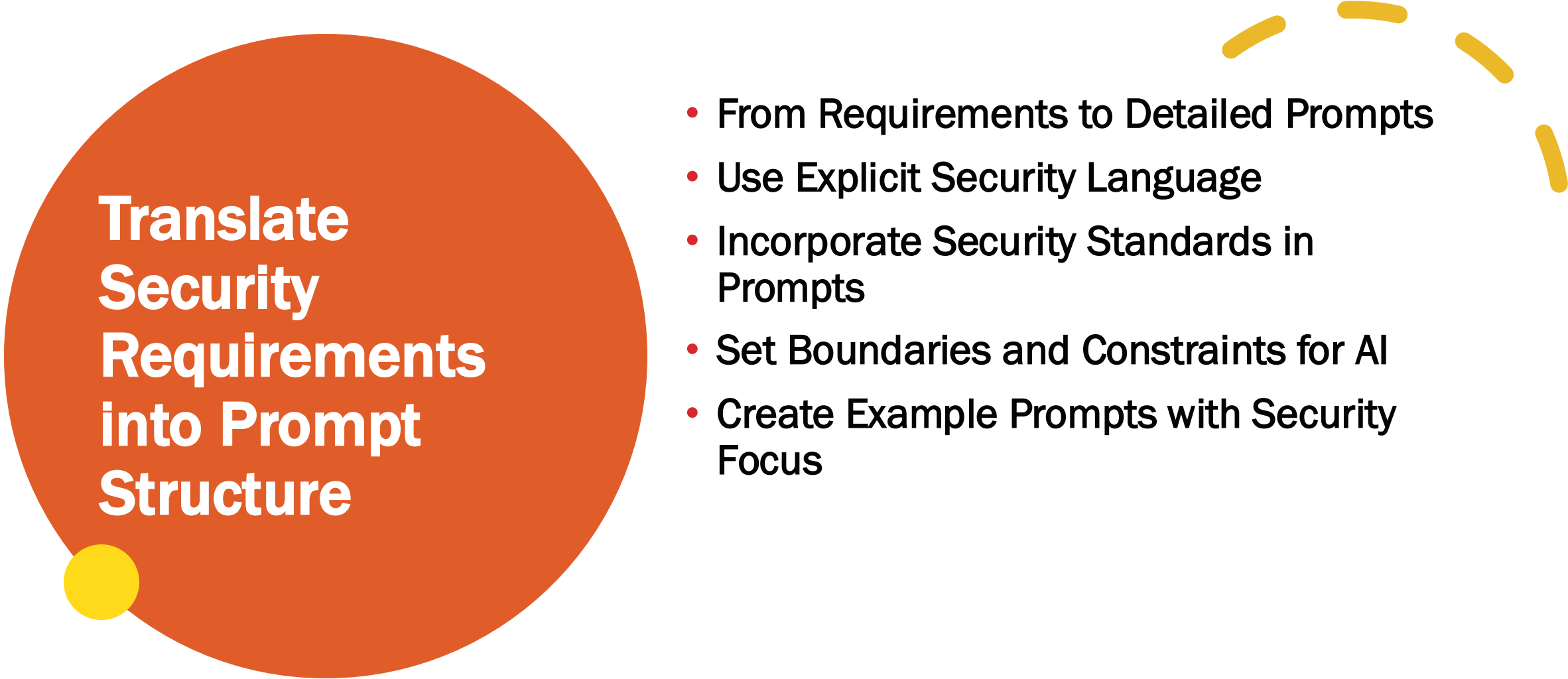


ITERATE AND REFINE YOUR
RULES



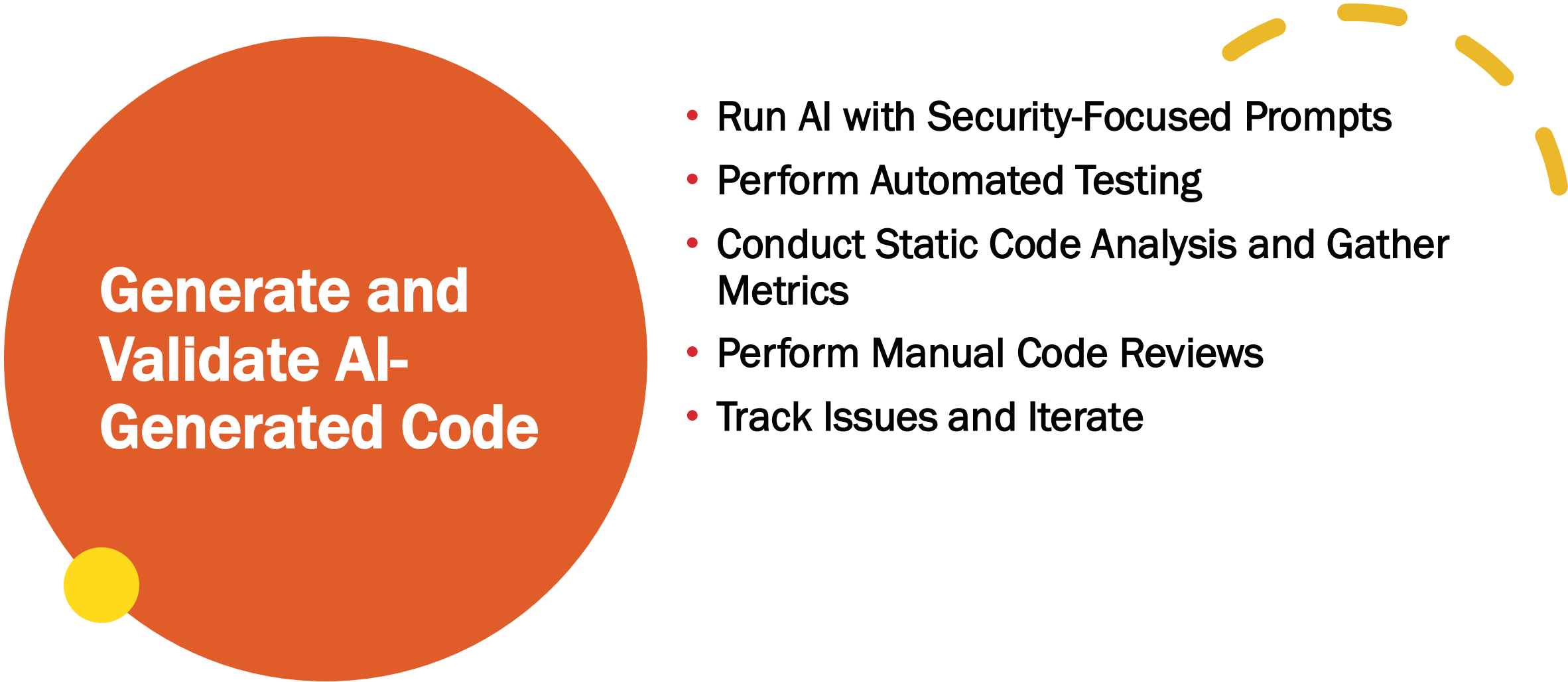
Identify Security Requirements

- Understand the Application's Security Context
- Establish Core Security Principles
- List Industry-Specific Security Standards
- Focus on Common Security Vulnerabilities
- Define Security Outcomes for AI Prompts



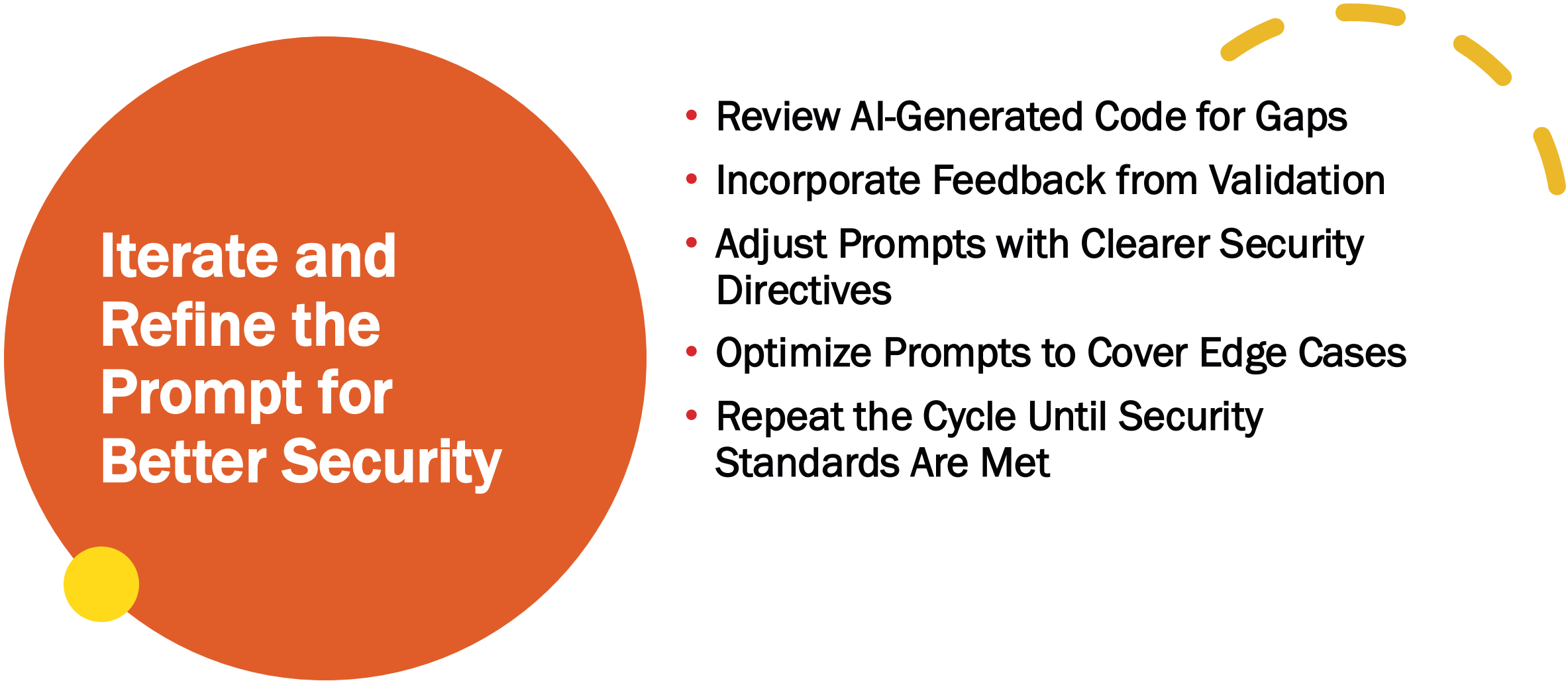
Translate Security Requirements into Prompt Structure

- From Requirements to Detailed Prompts
- Use Explicit Security Language
- Incorporate Security Standards in Prompts
- Set Boundaries and Constraints for AI
- Create Example Prompts with Security Focus



Generate and Validate AI- Generated Code

- **Run AI with Security-Focused Prompts**
- **Perform Automated Testing**
- **Conduct Static Code Analysis and Gather Metrics**
- **Perform Manual Code Reviews**
- **Track Issues and Iterate**



Iterate and Refine the Prompt for Better Security

- Review AI-Generated Code for Gaps
- Incorporate Feedback from Validation
- Adjust Prompts with Clearer Security Directives
- Optimize Prompts to Cover Edge Cases
- Repeat the Cycle Until Security Standards Are Met

AI Secure Code Generation Best Practices

Additional Prompt Best Practices

- Be Specific
- Explain Context and Purpose
- Include Constraints
- Provide Good Code Examples
- Specify Security Considerations
- Ask for Explanations
- Iterate Slowly and Refine
- Request Best Practices
- Put your prompts under version control

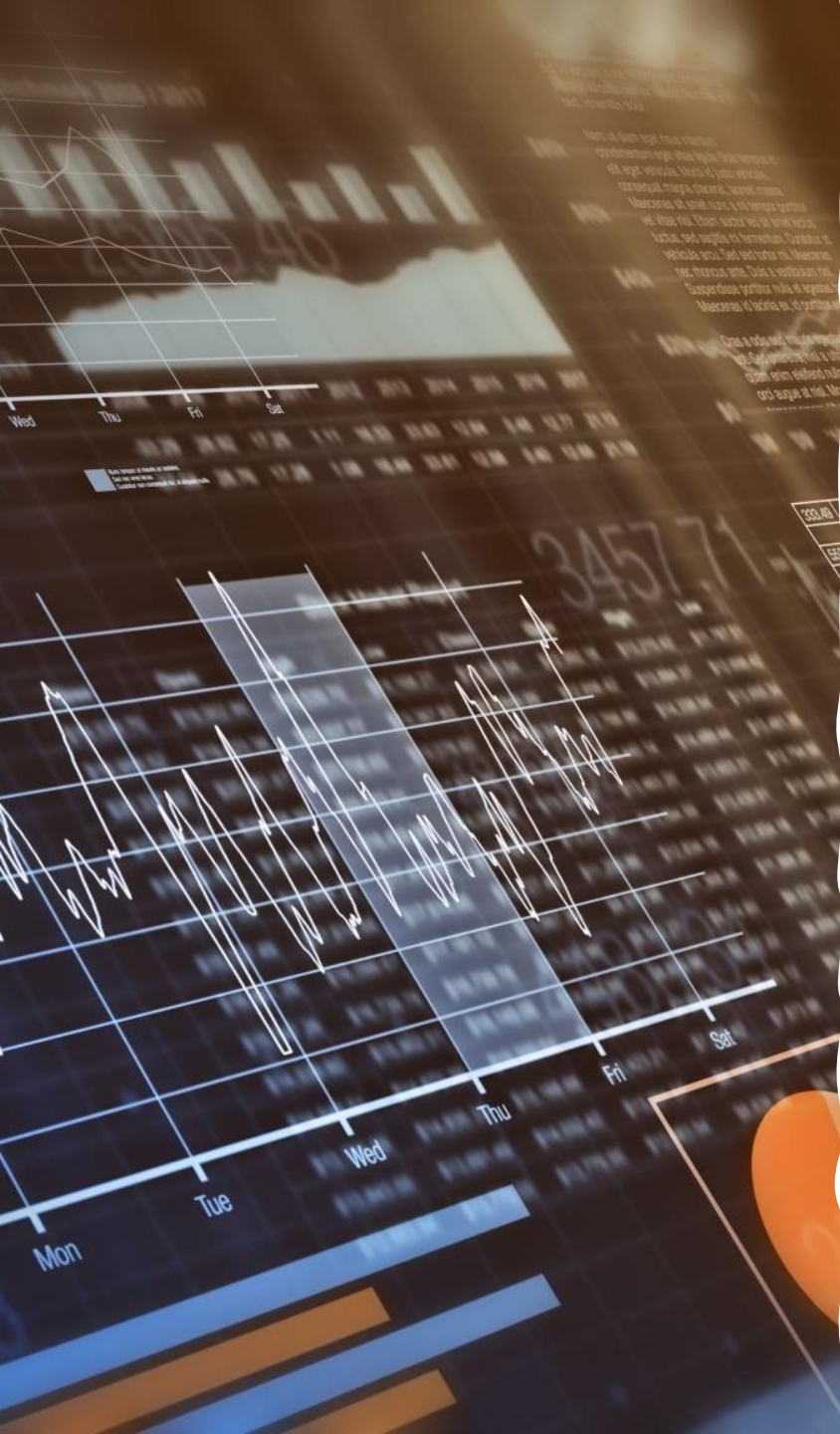




AI Code Generation

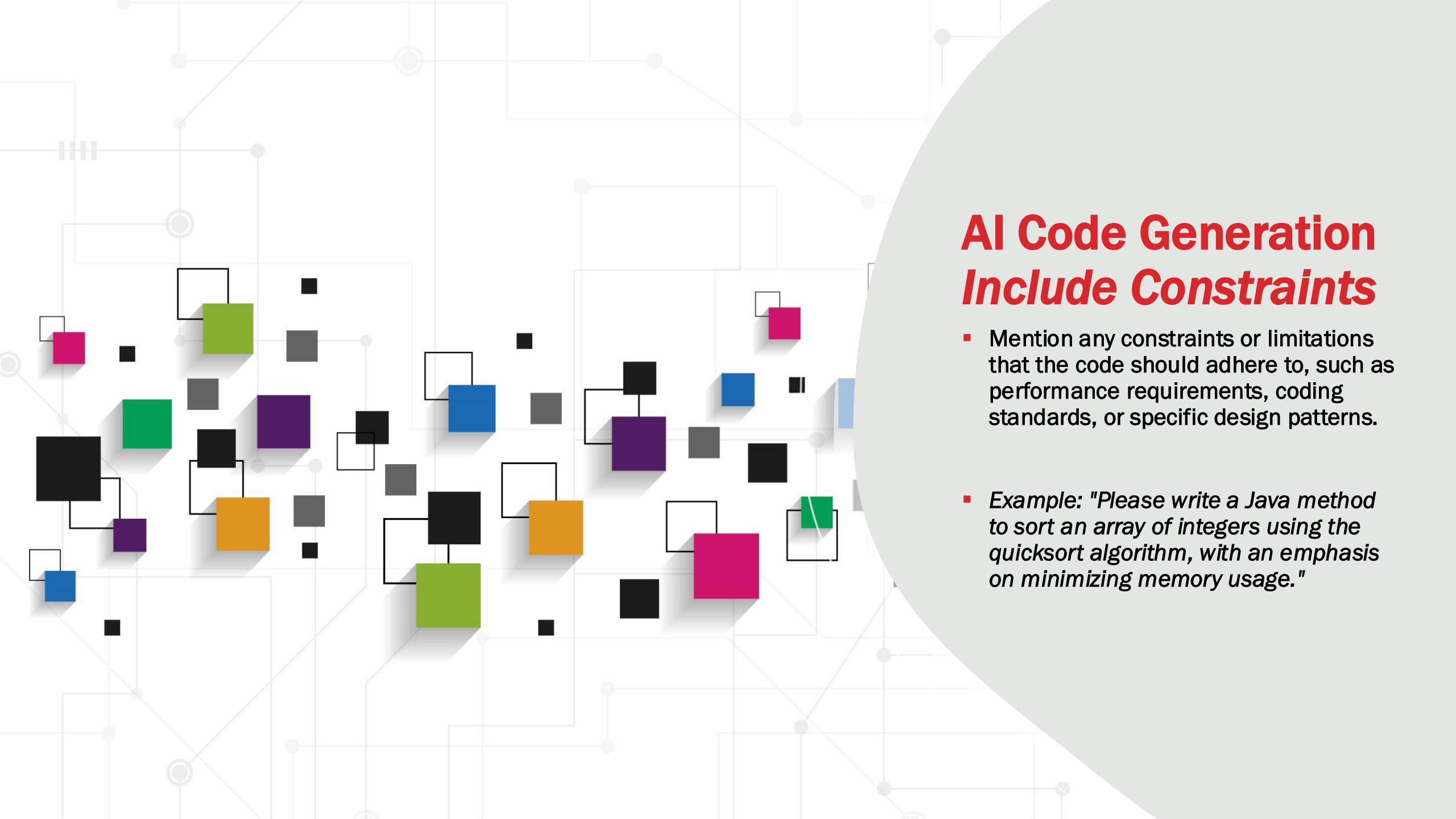
Be Specific

- Provide clear and detailed requirements for the code you need. Specify the language, functionality, and any libraries or frameworks to be used.
- *Example: "Can you create a Python script that reads a CSV file and outputs the average value of a specific column using pandas?"*



AI Code Generation *Context and Purpose*

- Explain the context or the purpose of the code. This helps the model understand how to tailor the solution.
- *Example: I need a Python function to parameterize user inputs to prevent SQL injection in a web application.*



AI Code Generation *Include Constraints*

- Mention any constraints or limitations that the code should adhere to, such as performance requirements, coding standards, or specific design patterns.
- *Example: "Please write a Java method to sort an array of integers using the quicksort algorithm, with an emphasis on minimizing memory usage."*

AI Code Generation Specify Security Considerations

- Explicitly state the security measures you want to be included in the code.
- *Example: "Can you write a Node.js function to hash passwords using bcrypt with a work factor of 11 for secure storage in a database?"*

AI Code Generation

Ask for Explanations

- Request explanations or comments in the code to understand the logic and flow better.
- *Example: "Could you write a JavaScript function to validate an email address and add comments explaining each step?"*



AI Code Generation

Iterate in Small Steps and Refine

- If the initial response is not perfect, provide feedback and ask for refinements or modifications.
- *Example: "The code you provided is good, but can you modify it to handle edge cases where the input list is empty?"*





AI Code Generation *Request Best Practices*

- Ask for best practices to be followed, such as code readability, efficiency, and maintainability.
- *Example: "Write a C# class for managing user sessions, following SOLID principles."*

Put your prompts under version control

- **Consistency and Reproducibility**
 - Track changes and maintain consistency across different versions.
 - Ensure reproducibility by using specific versions of prompts.
- **Collaboration and Sharing**
 - Facilitate collaboration among team members.
 - Easily share prompt updates and improvements.
- **History and Auditing**
 - Maintain a history of changes for auditing purposes.
 - Roll back to previous versions if needed.

- **Quality and Improvement**
 - Identify and fix issues in prompts over time.
 - Improve the quality of prompts through iterative changes.
- **Integration with CI/CD Pipelines**
 - Integrate prompt management with continuous integration/continuous deployment workflows.
 - Automate testing and deployment of prompts.
- **Documentation and Commenting**
 - Add comments and documentation to explain prompt changes.
 - Enhance understanding and transparency of prompt evolution.

AI Secure Code Generation Metrics

Importance of AI Code Evaluation Metrics



Supports Compliance and Standards



Ensures Consistent Performance



Identifies Security Gaps



Mitigates Risk of Failure



Improves Trustworthiness



Facilitates Maintenance



Helps Measure Code Quality

Categories of Metrics

AI Risk Lense View



Secure

Secure Coding Metrics

- Least Principle, Input Validation, Output Handling



Code

Code Maintainability Metrics

- Cyclomatic Complexity, Cognitive Complexity, High Cohesion



Run

Code Operational Quality Metrics

- Mean Time Between Failure (MTBF), Mean Time to Repair (MTTR), Runtime Availability

Secure Design Metrics	Evaluate security of software			
Identify Vulnerabilities	Report Risk Areas	Effectiveness Security Controls	Security Integrated Design	
Least Principle	Core access control principle based on business logic			
Input Validation	Validate that all inputs follow clear rules			
Output Handling	Use of all data is done safely.			

Code Reliability Metrics

Assess software ability to perform consistently

Expected Performance

Aim for Smooth Operation

Reduce Complexity

Achieve Robust Code

Cyclomatic Complexity

Measures the number of linearly independent paths through the code

Cognitive Complexity

Measures code understandability by humans

High Cohesion

Indicates how well-related the responsibilities of a module or class are

Runtime Metrics

Investigate performance and stability

**Identify
Improvements**

**Ensure
Robust Code**

**Ensure
Performance**

**Less Error
Prone Code**

MTBF

Measures the average time between system failures

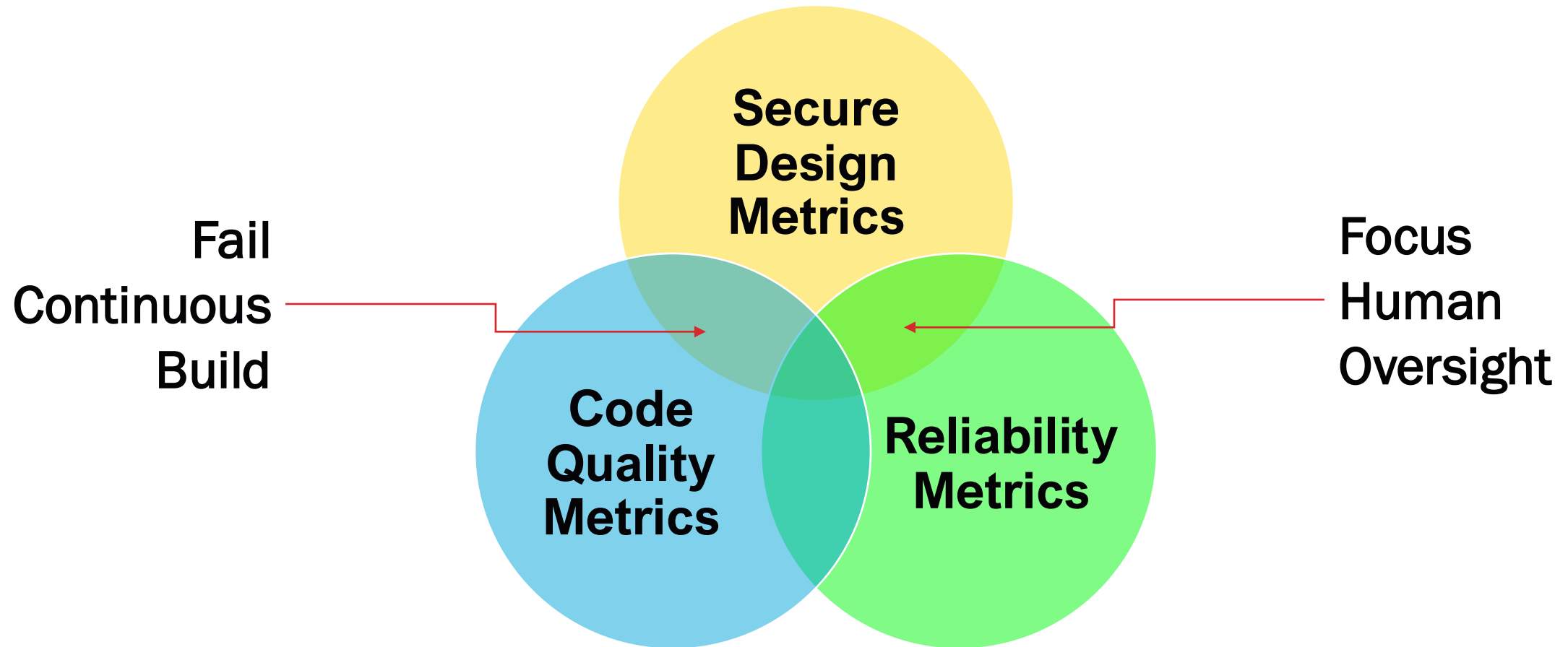
MTTR

Measures the average time it takes to repair a system after a failure

Availability

Refers to the percentage of time that a system is operational

How AI Code Evaluation Metrics Overlap



React Demo

Problem Statement

Language models, such as GPT, can write code from raw prompts.

Issue: They lack inherent knowledge of secure coding practices, which must be explicitly integrated.

Without human intervention or secure guidelines, AI might generate insecure code.

Key Secure Coding Principles in React

- Protect against XSS (Cross-Site Scripting)
- Sanitize and validate user inputs
- Client data and functionality minimization
- Ensure principle of least privilege in visible and hidden code
- Proper client-side authentication workflows
- Proper handling of edge-cases like user driven URL's rendered in React

Secure vs Insecure Prompting

- Original Prompt:
 - Generate a React login form."
- Secure Prompt:
 - Generate a React login form that sanitizes user inputs, validates input lengths and characters, includes CSRF tokens, and does not use dangerouslySetInnerHTML.

Converting Human Language into Secure Code Mechanisms

- Human Language Requirement
 - The form should be secure against XSS.
- Code Mechanism
 - Ensure no insecure usage of `dangerouslySetInnerHTML`, and use proper data binding and escaping.
- Example Prompt:
 - *Generate a React login form with validation, CSRF protection, and secure input handling against XSS. When using `dangerouslySetInnerHTML`, please sanitize that data with `DOMPurify`.*

Today's AI Code Generation Engines

GitHub Copilot - Code Completion Tool	AI-powered tool that generates code snippets and context-aware suggestions.
Risk Reported	May suggest hard-coding sensitive information like API keys or passwords.
Claude 3.5 - AI Model	Advanced AI model for coding, offering debugging and code generation.
Risk Reported	May produce biased or incorrect code due to limitations in training data.

ChatGPT 4o - Conversational AI Model	OpenAI's latest model enhances coding efficiency with interactive code generation and detailed explanations.
Risk Reported	It can produce verbose outputs that may confuse developers and lead to coding inefficiencies.
Gemini - AI Model	Google's AI model integrates application functionalities and offers coding support using Google's extensive data resources.
Risk Reported	As a newer model, it may lack robustness and accuracy, leading to incomplete or incorrect code suggestions.

Cody AI - Code Assistant	Analyzes codebases and assists with navigation and debugging.
Risk Reported	May generate code that doesn't follow best practices, leading to long-term maintainability issues.
Tabnine - Code Completion Tool	Focuses on security and compliance, providing code completion and intelligent suggestions.
Risk Reported	May suggest outdated or vulnerable code due to reliance on historical data.

Sourcery - Code Quality Tool	Enhances code quality by identifying errors and suggesting real-time improvements.
Risk Reported	May miss edge cases or context-specific issues, possibly introducing bugs if not carefully reviewed.
Rewind.ai - Documentation Tool	Automatically generates and updates documentation from code.
Risk Reported	The documentation may lack clarity or accuracy, leading to potential misunderstandings of the code.

CodeWhisperer (Amazon) - Code Completion Tool	Provides code suggestions, especially useful for AWS developers.
Risk Reported	May suggest code that's not optimized for performance or scalability in complex cloud environments.
Mutable AI - Code Assistant	Focuses on eliminating boilerplate and speeding up code delivery with intelligent suggestions.
Risk Reported	May introduce errors by removing essential boilerplate, potentially causing runtime issues.

AI Code Generation Conclusion

Challenges and Solutions in AI Code Generation

- **Challenge:** AI models do not naturally prioritize security.
- **Solution:** Explicitly include security requirements in the prompt and realize your prompt engineering security instructions are even more important than the code they generate.
- **Challenge:** Over-reliance on AI can lead to missed vulnerabilities.
- **Solution:** Always review (manual and automated) AI-generated code for compliance with security standards.



It's been a pleasure
jim@manicode.com

JIM MANICO | Secure Coding Instructor

www.manicode.com