Responsible AI with GitHub Copilot

Best practices for developers in the AI era

Presented by - Aditya

Introduction

What is Responsible AI?

The practice of designing, developing, and deploying AI with good intention to empower employees and businesses, and to fairly impact customers and society.

Copilot as a Pair Programmer

- ✓ AI is a tool to assist, not replace.
- ✓ Developers must remain the pilot in command.
- ✓ Always review and test AI-generated code.

Introduction

The Developer's New Reality

- AI coding assistants like GitHub Copilot are transforming software development speed and creativity.
- However, speed should not compromise safety.
- Developers are the "pilots" ensuring that AI-generated code is secure, fair, and functional.
- Responsible AI isn't just a policy; it's a daily coding practice.

Agenda

Mitigate AI Risks

Understanding potential pitfalls like hallucinations and bias.

6 Core Principles

Microsoft & GitHub's framework for ethical AI.

Practical Coding

Valid vs. Invalid patterns in C#, TS, and .NET.

Key Takeaways

Summary of best practices.

Mitigate AI Risks

Common Risks

- **Hallucinations:** AI suggesting libraries or APIs that don't exist.
- **Bias:** Generating code that reflects historical data biases.
- **Security:** Accidental exposure of secrets or use of vulnerable patterns.

The Mitigation Strategy

Human in the Loop: AI is the co-pilot; you are the pilot. Always review, test, and sanitize AI suggestions before merging.

Microsoft & GitHub's 6 Principles

These principles guide the development and usage of AI tools like Copilot.

- ✓ Fairness
- ✓ Reliability & Safety
- ✓ Privacy & Security
- ✓ Inclusiveness
- ✓ Transparency
- ✓ Accountability

Principles: Trust & Equity

Fairness

AI systems should treat all people fairly.

Dev Action: Check algorithms for bias against any group. Ensure training data is diverse.

Inclusiveness

AI should empower everyone and engage people.

Dev Action: Design accessible interfaces (WCAG) and consider users with different abilities.

Transparency

AI systems should be understandable.

Dev Action: Document how AI is used in your app. Explain limitations to users.

Principles: Safety & Integrity

Reliability & Safety

AI should perform reliably and safely.

Dev Action: Rigorous testing. Handle edge cases where AI fails gracefully.

Privacy & Security

Respect privacy and ensure security.

Dev Action: Never hardcode secrets. Don't send PII to public AI models.

Accountability

People should be accountable for AI systems.

Dev Action: Maintain human oversight. You are responsible for the code you ship.

C# Example: Managing Secrets

INVALID

```
public class AzureService {
    private string _apiKey = "AIzaSyD-...";
    // ❌ Risk: Hardcoding secrets exposes
    // keys in version control.

    public void Connect() { ... }
}
```

Never allow AI to autocomplete hardcoded credentials. Always review string assignments.

VALID

```
public class AzureService {
    private string _apiKey;

    public AzureService() {
        _apiKey =
Environment.GetEnvironmentVariable("AZURE_KEY");
        // ✅ Safe: Load from environment
        // or Key Vault.
    }
}
```

Use standard configuration patterns. Prompt Copilot to use "Environment Variables".

C# Example: **C# Security**

Scenario: Handling Database Queries. AI might suggest concatenating strings.

INVALID

```
ser GetUser(string username) {
    // ❌ VULNERABLE: Direct string concatenation
    string query = "SELECT * FROM Users WHERE Name = '"
                   + username + "'";

    using (var command = new SqlCommand(query, connection)) {
        // ... executes command
    }
}
```

AI might suggest this if context is simple string manipulation.

VALID

```
User GetUser(string username) {
    // ✅ SECURE: Using Parameters
    string query = "SELECT * FROM Users WHERE Name = @Name";

    using (var command = new SqlCommand(query, connection)) {
        command.Parameters.AddWithValue("@Name", username);
        // ... executes command
    }
}
```

Correct pattern: Always use parameterized queries.

C# Example: Security

Copilot might sometimes suggest hardcoding credentials if the context implies a quick test. Always refactor to use secure environment variables.

INVALID

```
const   aws_key = "AKIAIOSFODNN7EXAMPLE";
const   db_pass = "superSecret123"      ;
// Never commit secrets to repo!
```

VALID

```
const   aws_key = process.env.AWS_ACCESS_KEY;
const   db_pass = process.env.DB_PASSWORD;
// Load from .env file securely
```

Avoid Hardcoded Secrets

ASP.NET Core Example: Privacy

Scenario: Logging user actions for debugging.

INVALID

```
[HttpPost("login")]
public IActionResult Login(UserDto user) {
    // ❌ DANGEROUS: Logging entire object
    // This logs passwords and PII to plain text files
    _logger.LogInformation("Login attempt: {@User}", user);

    // ... logic
}
```

Violates Privacy principle by exposing sensitive data.

VALID

```
[HttpPost("login")]
public IActionResult Login(UserDto user) {
    // ✅ SAFE: Log only identifiers
    _logger.LogInformation("Login attempt for UserID: {Id}",
        user.Id);

    // ... logic
}
```

Respects user privacy and minimizes data risk.

ASP.NET Core: Authorization

INVALID

```
[AllowAnonymous]
public IActionResult DeleteUser(int id) {
    _repo.Delete(id);
    // ❌ Risk: Unprotected administrative
    // action suggested for "ease of testing".
    return Ok();
}
```

Be wary of AI removing security gates for convenience during scaffolding.

VALID

```
[Authorize(Roles = "Admin")]
public IActionResult DeleteUser(int id) {
    _repo.Delete(id);
    // ✅ Safe: Explicit role-based
    // access control.
    return Ok();
}
```

Always verify that generated controllers have correct attributes like `[Authorize]`.

TypeScript Example: Input Validation

INVALID

```
const processInput = (data: any) => {
    eval(data.expression);
    // ❌ Risk: 'eval' is dangerous and
    // opens up injection attacks.
};
```

AI might suggest quick hacks like `eval` or `innerHTML`. Reject these immediately.

VALID

```
const processInput = (data: string) => {
    const sanitized = sanitize(data);
    const result = safeParse(sanitized);
    // ✅ Safe: Validate and sanitize
    // all external inputs.
    return result;
};
```

Enforce strict typing and sanitization. Use established libraries over ad-hoc regex.

Key Takeaways

Checklist for Responsible Devs

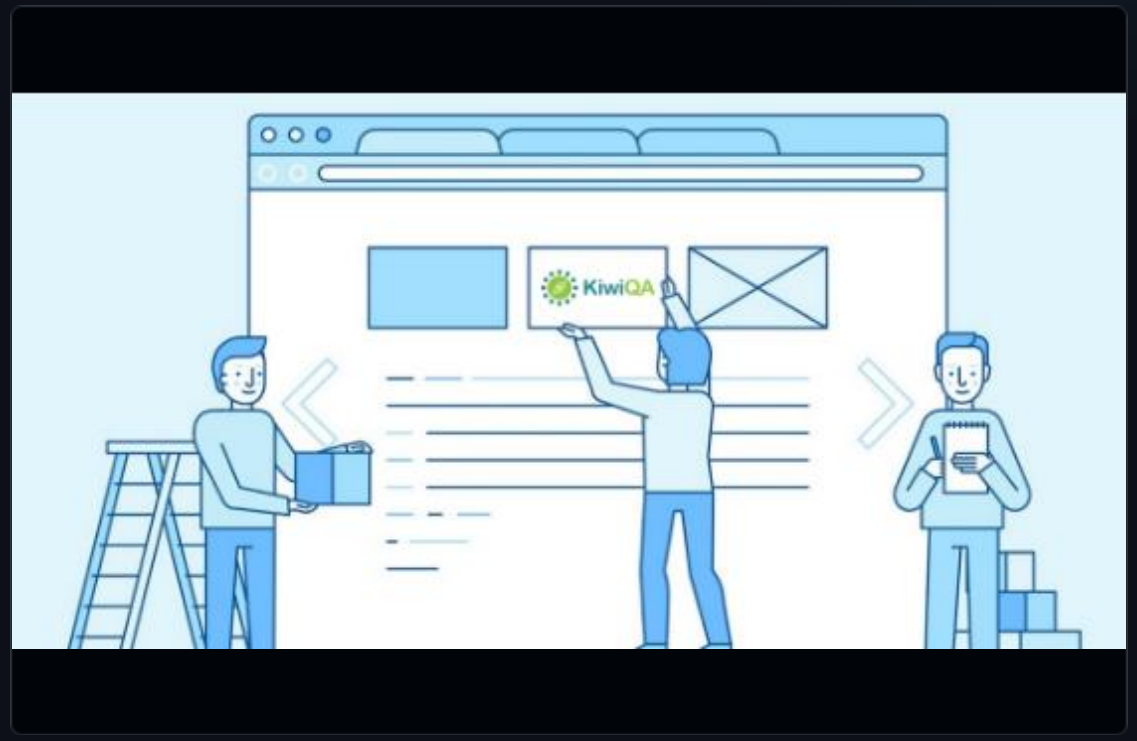
- ✓ **Review Everything:** Treat AI code as a suggestion, not a solution.
- ✓ **Scan for Secrets:** Ensure no API keys enter the codebase.
- ✓ **Test Rigorously:** Unit tests are more important than ever.
- ✓ **Stay Secure:** Apply standard security principles (OWASP) to AI code.

Key Takeaways

- **Review Everything:** Treat AI code as a suggestion, not a solution.
- **Context Matters:** You understand the business logic and ethics; AI predicts the next token.
- **Apply Principles:** Check for bias, security flaws, and accessibility in every PR.
- **Stay Secure:** Never let AI handle secrets or PII without sanitization.

Best Practices for Developers

- **Review Output:** Treat AI code like code from a junior developer—review it thoroughly.
- **Context is King:** Keep open tabs relevant. Copilot uses open tabs for context; close unrelated files to reduce hallucinations.
- **Use Filters:** Enable GitHub's public code filter to avoid potential IP issues.
- **Stay Updated:** AI models evolve. Keep your IDE extension updated for the latest security patches.



Q & A

Thank You!