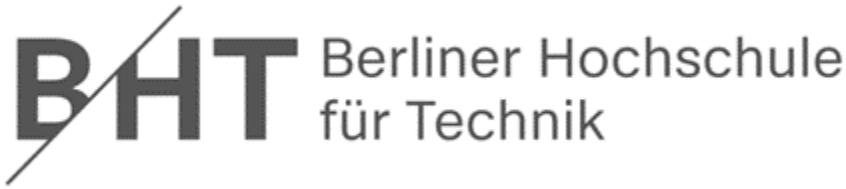
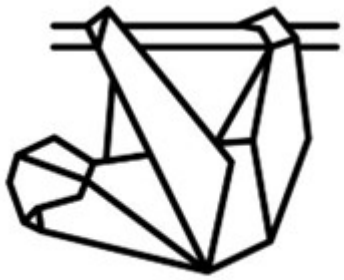


Fachbereich VI - Informatik und Medien
Studiengang IT-Sicherheit Online / Medieninformatik

Vorbereitung 8 - Bedrohungsmodellierung, Entwicklungsprozess



PARETO CONSULTING

Modul:	Sicherheitsmanagement
Dozent:	Sven Zehl
vorgelegt von:	Gruppe 1 Christine Kuczera Dirk Drutschmann Hicham Naoufal Michael Schröter Jan Zimmermann Ivo Valls

Aufgabe 1) Kennen Sie Grundsätze guten Programmierens?

Antwort

KISS-Prinzip (Keep It Simple, Stupid)

Dieses Prinzip besagt, dass Code so einfach wie möglich gehalten werden sollte. Es geht darum, Komplexität zu vermeiden und unnötige Abstraktionen oder Funktionalitäten zu vermeiden. Einfacher Code ist leichter zu verstehen, zu warten und zu erweitern.

DRY-Prinzip (Don't Repeat Yourself)

Dieses Prinzip besagt, dass Code nicht redundant sein sollte. Wiederholungen von Code führen zu einer erhöhten Wartungsarbeit und einem höheren Risiko von Fehlern. Stattdessen sollte Code so organisiert werden, dass gemeinsame Funktionalitäten in wiederverwendbaren Modulen oder Funktionen gekapselt werden.

SOLID-Prinzipien

SOLID ist ein Akronym für eine Reihe von Prinzipien, die das Design von Softwarekomponenten unterstützen:

Single Responsibility Principle (SRP):

Eine Klasse sollte nur eine einzige Verantwortung haben.

Open/Closed Principle (OCP):

Klassen sollten für Erweiterungen offen, aber für Modifikationen geschlossen sein.

Liskov Substitution Principle (LSP):

Objekte einer abgeleiteten Klasse sollten sich anstelle von Objekten der Basisklasse substituieren lassen, ohne dass die korrekte Funktionalität beeinträchtigt wird.

Interface Segregation Principle (ISP):

Schnittstellen sollten spezifisch für die Bedürfnisse der Klienten sein, um eine Kopplung zu vermeiden.

Dependency Inversion Principle (DIP):

Abhängigkeiten sollten von abstrakten Klassen oder Schnittstellen abhängen, nicht von konkreten Implementierungen.

Diese Prinzipien fördern die Trennung von Verantwortlichkeiten, Flexibilität, Erweiterbarkeit und Testbarkeit von Code.

YAGNI-Prinzip (You Ain't Gonna Need It)

Dieses Prinzip besagt, dass Code nur implementiert werden sollte, wenn er für die aktuelle Anforderung benötigt wird. Es ist eine Warnung vor übermäßiger Vorausplanung und dem Hinzufügen von Funktionen,

die möglicherweise nie verwendet werden. Das Ziel ist es, sich auf die unmittelbaren Anforderungen zu konzentrieren und den Code schlank und einfach zu halten.

TDD (Test-driven Development)

TDD ist eine Entwicklungspraktik, bei der Tests vor dem eigentlichen Code geschrieben werden. Der Zyklus besteht aus drei Schritten: Schreiben eines Tests, Implementierung des Codes, um den Test zu bestehen, und anschließende Refaktorisierung des Codes. TDD fördert eine hohe Testabdeckung und ermöglicht eine iterative Entwicklung, bei der der Code ständig getestet und überarbeitet wird, um die Funktionalität zu gewährleisten.

Dokumentation

Eine gute Dokumentation ist entscheidend, um anderen Entwicklern zu ermöglichen, den Code zu verstehen und damit zu arbeiten. Sie kann in verschiedenen Formen vorliegen, einschließlich Inline-Kommentaren, technischer Dokumentation, Benutzerhandbüchern oder Readme-Dateien. Die Dokumentation sollte den Code erklären, wichtige Designentscheidungen erläutern, API-Referenzen bereitstellen und Anleitungen für Entwickler und Benutzer bieten. Eine gut dokumentierte Codebasis erleichtert anderen Entwicklern den Einstieg und die Zusammenarbeit an einem Projekt.

Code-Reviews

Code-Reviews sind ein Prozess, bei dem Entwickler den Code ihrer Kollegen überprüfen und Feedback geben. Durch Code-Reviews können potenzielle Fehler, schlechte Praktiken oder Verbesserungsmöglichkeiten identifiziert werden. Dieser Prozess fördert die Qualität, Lesbarkeit und Korrektheit des Codes. Entwickler können voneinander lernen, bewährte Methoden teilen und sicherstellen, dass der Code den Standards und Prinzipien des guten Programmierens entspricht. Code-Reviews ermöglichen es auch, Fehler frühzeitig zu erkennen und zu beheben, bevor sie in die Produktion gelangen.

Aufgabe 2) Was ist "CVE"?

Antwort

CVE steht für Common Vulnerabilities and Exposures, was übersetzt so viel wie "gemeinsame Schwachstellen und Offenlegungen" bedeutet. CVE ist ein Standard zur eindeutigen Identifizierung von Sicherheitslücken und -bedrohungen in Software und Hardware. Es handelt sich um eine öffentliche Sammlung von Informationen zu Sicherheitslücken, die von der Community verwaltet wird.

Jede Sicherheitslücke, die im CVE-System erfasst wird, erhält eine eindeutige Kennung in Form von "CVE-YYYY-NNNN", wobei "YYYY" das Jahr und "NNNN" eine fortlaufende Nummer darstellt. Diese Kennung ermöglicht es, spezifische Sicherheitslücken eindeutig zu identifizieren und darüber zu kommunizieren.

Die CVE-Liste dient als Referenz für Unternehmen, Organisationen, Entwickler und Sicherheitsexperten, um über bekannte Schwachstellen informiert zu sein und geeignete Maßnahmen zum Schutz ihrer Systeme zu ergreifen. Sie wird von vielen Sicherheitsdienstleistern, Herstellern und Sicherheitsbehörden weltweit genutzt.

Es ist wichtig anzumerken, dass CVE lediglich eine eindeutige Identifizierung und Referenzierung von Sicherheitslücken bietet. Die eigentliche detaillierte Beschreibung der Schwachstelle und mögliche Lösungen finden sich oft in den sogenannten CVE-Einträgen oder in den entsprechenden Sicherheitsbulletins der betroffenen Hersteller oder Projekte

Aufgabe 3) Im Rollenspiel Kosten der Sicherheit läuft gerade die Entwicklung aus dem Ruder, der Auslieferungstermin wird nicht zu halten sein. – Warum wird das Projektrisiko meist zu optimistisch angesetzt? Analysieren Sie den Spielraum des Scrum Product Owners gegenüber seinen Entwicklern und gegenüber der Geschäftsführung!

Siehe: https://hetzlefetz.github.io/uni-sicherheitsmanagement/chat/dist/index.html