Clean Code & Design Pattern

Tag 3

Zeitplan

9:00

Start

Pause

10:15 & 11:45

13:00

Ende

Broken-Windows-Principle

- “Wird eine zerbrochene Fensterscheibe nicht schnell repariert, sind im Haus bald alle Scheiben zerbrochen.”
- Abwärtsspirale der Qualität

→ KPI: Code Smells (statische Codeanalyse)

Review Tag 2

Version Control System

- Protokollierung der Änderungen
- Wiederherstellung einzelner Dateien
- Archivierung von Projektständen (+Tags/Labels)
- Gemeinsames arbeiten am Code durch Änderungszusammenführung
- Losgelöstes arbeiten am Code durch Verzweigung

First, be clear that Git is not GitHub. Git is a version control system in which you create a repository and commit changes one at a time. GitHub is a website that hosts your Git repository. GitHub's UI only supports creation and editing of text files, so if you need binaries, you need to use Git directly.

Clean Code

- Single Responsibility Principle
- Separation of Concerns
- Dependency Inversion Principle
- “Think Big, Start Small”
- “…im Moment, erst einmal, reicht nur…”

Design Pattern

- Strategy Pattern (Logging)
- Command Pattern
- Composite Pattern
    a. BatchCommand fasst Commands zusammen
    b. BatchCommand wird wie ein atomares Command behandelt

Dependency Injection Libraries/Frameworks

- Erstellt die Instanzen (für Interfaces oder Klassen)
- Erstellt die Abhängigkeiten
- Verantwortlich für den “Lifecycle”

→ Erledigt die lästige Tipparbeit

Code auslagern

- Methode → Parameter einführen
- Klasse → Interface einführen
- Projekt (csproj) → Factory einführen
- Solution (sln) → nuget einführen

Creational Design Pattern

…und die Herstellung von Objekten

Clean Code Prinzipien (Auszug)

- KISS → Keep It Simple and Stupid
- YAGNI → You Ain't Gonna Need It
- Don't optimize prematurely (applies to performance, structure, and design pattern)
  - The first rule of optimization is: Don't do it.
  - The second rule of optimization (for experts only) is: Don't do it yet.

Abhaengigkeiten - “New is Glue”

- New() kennt die Implementierung
- New() kennt den Lebenszyklus (Lifecycle)
- New() kennt die Konstruktor Anhängigkeiten

Dilemma bei DI: Keine Kontrolle über die Erstellung

→ Factory Pattern/Abstract Factory

→ Service Locator

Factory Method

Factory Method

- Ein Objekt wird durch den Aufruf einer Methode erzeugt,

  anstatt durch den direkten Aufruf des Konstrukturs
- Aufrufer kennt die Abhängigkeiten und den Instanziierungsprozess nicht
- Trennung zwischen Erstellung und Funktion

→ Ich kenne jemanden, der das kann

→ Z.B. Privater Konstruktor

Demo

Factory Method implementieren

Projekt: “PdfTools”

NLogFactory implementieren
Interface ILoggerFactory
Rückgabe ILogger

Refactoring

Refactoring

- “Refactoring […] bezeichnet […] die manuelle oder automatisierte **Strukturverbesserung** […] unter **Beibehaltung** des *beobachtbaren* **Programmverhaltens**” –Wikipedia

- Keine Veränderung des äußeren Verhaltens
- Veränderung des inneren Verhaltens
  - Performance
  - Erweiterbarkeit, Austauschbarkeit
  - Testbarkeit, Struktur

Complex Refactorings

- “Es ist nicht möglich, Code direkt in der ultimativen Form zu schreiben.”
- Irgendwann sind größere Eingriffe erforderlich
- Komplexe Refactorisierung benötigt Unit Tests

Zero-Impact-Injection Pattern (ZIIP)

- Klassen-internes Refactoring
  - Extract Class
  - Dependency Injection
  - Default-Parameter (null)
  - null-coalescing operator (??)
- Ergebnis: Code außerhalb der Klasse bleibt bestehen

Demo

Code Generator als ZIIP

Projekt “PdfTools”

Code Generator injizieren

Resuemee

- Lesbarer Code durch klarere Strukturen (DRY, SRP, SLAP, IOSP)
- Testbarer Code durch DIP (Wrapper, ZIIP, SRP)
- Robuster Code durch bessere Testbarkeit (Mock, Exceptions, Standardverhalten)
- Dokumentierter Code durch Desgin Pattern
- Qualitativer Code durch Kaizen (Pfadfinderregel, Refactoring, Broken-Window-Principle)