Refactoring und Legacy Code

Bestehenden Code kontrolliert verbessern, ohne sein Verhalten zu verändern

01 INTRO

Refactoring ist mehr als das Aufräumen von Code. Im eigentlichen Sinne handelt es sich um eine kontrollierte Veränderung seiner inneren Struktur, während das von außen beobachtbare Verhalten erhalten bleibt. Gerade bei Legacy Code entscheidet die Absicherung dieser Verhaltenserhaltung darüber, ob Verbesserungen kontrollierbar oder riskant werden.

Der Kurs vermittelt Refactoring deshalb als systematische Engineering-Technik. Die Teilnehmer erkennen Code Smells, schaffen bei Bedarf zunächst ein Sicherheitsnetz und führen nachvollziehbare kleine Transformationen durch. Dabei wird klar zwischen Refactoring, Fehlerkorrektur und funktionaler Änderung unterschieden.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Praktische Programmiererfahrung und grundlegende Kenntnisse automatisierter Tests

02 TRAININGSPROGRAMM

Inhalte

Refactoring exakt einordnen

- Definition
- Verhaltenserhaltung
- interne versus externe Struktur
- Refactoring versus Feature
- Refactoring versus Bugfix
- Refactoring versus Rewrite
- kontrollierte Transformation

Warum Code schwer änderbar wird

- gewachsene Strukturen
- Zeitdruck
- fehlende Tests
- wechselnde Anforderungen
- technische Schulden
- Wissen im Code
- Erosion

Code Smells als Hinweise

- Duplication
- Long Method
- Large Class
- Feature Envy
- Data Clumps
- Primitive Obsession
- Shotgun Surgery
- Message Chains

Sicherheitsnetz vor Veränderungen

- vorhandene Tests
- fehlende Tests
- Characterization Tests
- Golden Master
- Schnittstellen beobachten
- Verhalten erfassen
- Risiken priorisieren

Kleine Refactorings

- Rename
- Extract Method
- Inline Method
- Extract Variable
- Introduce Parameter Object
- Move Method
- Move Field
- Encapsulate Variable

Verantwortlichkeiten verändern

- Extract Class
- Inline Class
- Delegation
- Kohäsion erhöhen
- Kopplung reduzieren
- Schnittstellen
- Abhängigkeiten

Bedingungen und Logik verbessern

- Decompose Conditional
- Guard Clauses
- Replace Conditional with Polymorphism
- komplexe Bedingungen
- Null-Behandlung
- Kontrollfluss
- Seiteneffekte

Abhängigkeiten aufbrechen

- Dependency Injection
- Seams
- Wrapper
- Adapter
- externe Systeme
- Datenbanken
- statische Abhängigkeiten
- Testbarkeit herstellen

Refactoring regelgerecht durchführen

- kleiner Schritt
- Test
- Transformation
- erneuter Test
- Verhalten beobachten
- Änderungen isolieren
- jederzeit lauffähiger Zustand
- nachvollziehbare Commits

Werkzeuge nutzen

- IDE-Refactorings
- statische Analyse
- Code Search
- Coverage
- Versionsverwaltung
- Diff
- automatisierte Tests
- Werkzeuggrenzen

Refactoring im Entwicklungsprozess

- vorbereitendes Refactoring
- kontinuierliches Refactoring
- Boy Scout Rule
- Refactoring vor Features
- Code Reviews
- Branching
- Integration
- Refactoring-Budget

Legacy-Code-Übung

- unbekannten Code verstehen
- Risiken identifizieren
- Verhalten absichern
- Smells auswählen
- schrittweise refactoren
- Tests wiederholt ausführen
- Ergebnis bewerten

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	3 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams
Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner mit Entwicklungsumgebung für die praktischen Übungen

> **Inhouse-Schulungen**
> Ein Ausschnitt aus dem eigenen Legacy-System kann als Grundlage für die Übungen dienen.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Softwareentwickler und technische Quality Engineers, die bestehenden oder schwer wartbaren Code sicher weiterentwickeln müssen.

> **Voraussetzungen**
> Praktische Programmiererfahrung und Grundlagen automatisierter Tests.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- Refactoring eindeutig von funktionalen Änderungen unterscheiden
- Legacy Code vor Änderungen charakterisieren
- geeignete Refactorings auswählen
- Veränderungen in kleinen kontrollierten Schritten durchführen
- Tests als Sicherheitsnetz einsetzen
- Abhängigkeiten schrittweise aufbrechen
- Refactoring in den normalen Entwicklungsprozess integrieren

06 FAQ

Häufige Fragen

Muss Legacy Code zunächst vollständig mit Tests abgedeckt werden?

Nein. Häufig wäre das wirtschaftlich unmöglich. Entscheidend ist eine risikoorientierte Absicherung der Bereiche, die verändert werden.

Warum sind kleine Refactoring-Schritte so wichtig?

Sie begrenzen das Risiko und erleichtern die Zuordnung eines Fehlers zu einer konkreten Änderung. Nach jedem Schritt kann überprüft werden, ob das Verhalten erhalten geblieben ist.

Wird auch ein kompletter Rewrite behandelt?

Als Alternative wird er diskutiert. Ein Rewrite ist aber kein Refactoring.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/refactoring-legacy-code/