HTWK Hochschule für Technik, Wirtschaft und Kultur Leipzig

Refactoring und technische Schulden in agilen Projekten

8 JANUAR, 2024

Felix Herrling, Oliver Lindemann, Tim Dietrich

Gliederung

1. **Technische Schulden**
2. **Refactoring**
3. **Fazit**
4. **Diskussion**
5. **Quellen**

Technische Schulden
Begriffsklärung

- Rückstände in der Software während der Entwicklung, welche die zukünftige Produktivität negativ beeinflussen
- Metapher zu Schulden in der Finanzwelt
- soll den Grund für steigende Kosten in der Softwareentwicklung verdeutlichen
- allgemein anerkannte Definition existiert nicht
- globale TS: 500 Milliarden US-Dollar (2010)

Technische Schulden
Begriffsursprung

- Erfahrungsbericht von Ward Cunningham (1992)
- Softwareprojekt der Firma “Wyatt Software”

“Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite.”

- Anpassung der Architektur vorerst nur für neue Features
- bewusstes Vorgehen
- Metapher für Kommunikation mit Projektleitung

Quelle: https://en.wikipedia.org/wiki/File:Ward_Cunningham_-_Commons-1.jpg

Technische Schulden
Vorteile

- Wettbewerbsvorteil
- schnelleres Kundenfeedback
- neue Erfahrungen
- bessere Entscheidungen
- höhere Qualität zukünftiger Arbeit

Technische Schulden
Gefahren

- Zinsen in Form von Mehrkosten / Mehraufwand

“The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.”

- Komplexität des Quellcodes steigt
- Wartbarkeit und Produktivität werden schlechter
- mehr Fehler und Missverständnisse

Technische Schulden
Entstehung

Technische Schulden
Entstehung

- verschiedene Ursachen für die Entstehung technischer Schulden
- vor allem rücksichtslose Schulden sind gefährlich
- vorsichtiges Verhalten reduziert Gefahren
- Technische Schulden entstehen immer

Technische Schulden
Kategorisierung

- Quellcode
- Architektur
- Tests
- Dokumentation
- Anforderungen
- Infrastruktur

Technische Schulden
Beispiel Code Smells

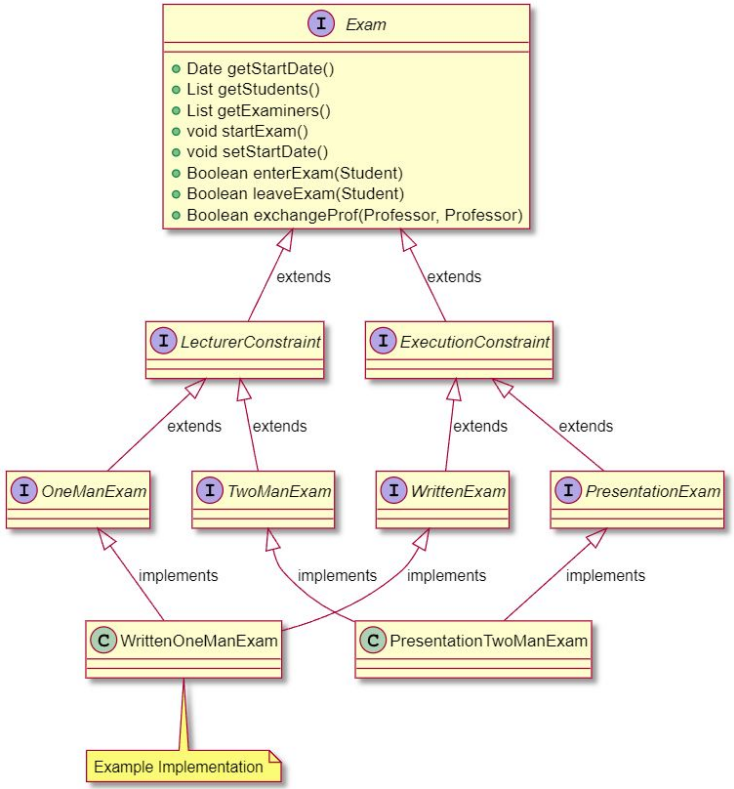
```
class Student {
    public String StudentName;
}
```

```
if (date.isBefore(plan.summerBreakStart) && date.isAfter(plan.summerBreakEnd)) {
    ticketFee = dayCount * plan.summerBreakFee + plan.regularCharge;
} else {
    ticketFee = dayCount * plan.semesterFee;
}
```

Technische Schulden
Beispiel Architektur

- Beispiel aus Seminar Übung 02
- Erweiterung neuer Exam-Typen soll durch Interface-Struktur ermöglicht werden

- Neue Anforderung in Übung 03
  - Attribute dürfen nach Start der Prüfung nicht mehr geändert werden
- schlecht umsetzbar in aktueller Architektur
- TS wird sichtbar ⇒ Lösung?



Technische Schulden
Beispiel Tests

```java
import java.time.Duration;
import java.time.LocalDateTime;

public class Timeslot {

    public LocalDateTime start;
    public LocalDateTime end;

    public Timeslot(LocalDateTime start, LocalDateTime end) {
        this.start = start;
        this.end = end;
    }

    public Duration getDuration() {
        return Duration.between(start, end);
    }
}
```

Technische Schulden
Beispiel Tests

```java
public class Timeslot {

    public LocalDateTime start;
    public LocalDateTime end;

    public Timeslot(LocalDateTime start, LocalDateTime end) {
        if (start == null || end == null) {
            throw new IllegalArgumentException("Start- und Endzeitpunkt dürfen nicht null sein."
        }
        this.start = start;
        this.end = end;
    }

    public Duration getDuration() {
        return Duration.between(start, end);
    }
}
```

Technische Schulden
Beispiel Tests

```
public class Timeslot {

    public LocalDateTime start;
    public LocalDateTime end;

    public Timeslot(LocalDateTime start, LocalDateTime end) {
        if (start == null || end == null) {
            throw new IllegalArgumentException("Start- und Endzeitpunkt dürfen nicht null sein."
        }

        if (!end.isAfter(start)) {
            throw new IllegalArgumentException("Endzeitpunkt muss nach dem Startzeitpunkt sein")
        }
        this.start = start;
        this.end = end;
    }
}
```

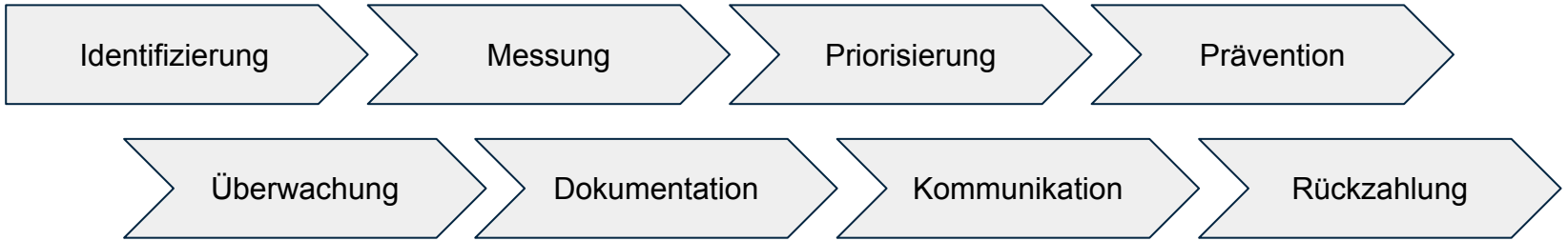
Technische Schulden
Eigenschaften

- Sichtbarkeit
  - Wie schnell/einfach kann die TS erkannt werden?
- Wert
  - Unterschied Zustand mit TS ⇒ idealer Zustand
- Verzinsung
  - Wie stark steigt der Wert der TS?
- Zeitwert
  - Wert + Verzinsung zum aktuellen Zeitpunkt
- Umfeld / Herkunft
  - entsprechend der Kategorisierung
- Einfluss
  - Wie viele / welche Komponenten sind betroffen?

Technisches Schuldenmanagement

- Ziele:
  - Prävention
  - Verwaltung und
  - Reduktion technischer Schulden

- Unterteilbar in Aktivitäten:



Technisches Schuldenmanagement
Identifikation

- Schulungen nötig
- Code-Schulden
  - Code-Reviews
  - automatisierte Tests
  - Code-Analysen
- Architektur-/ Design-Schulden
  - Indikatoren
    - Modularität
    - Gott-Klassen
    - Abhängigkeiten

Technisches Schuldenmanagement
Messung

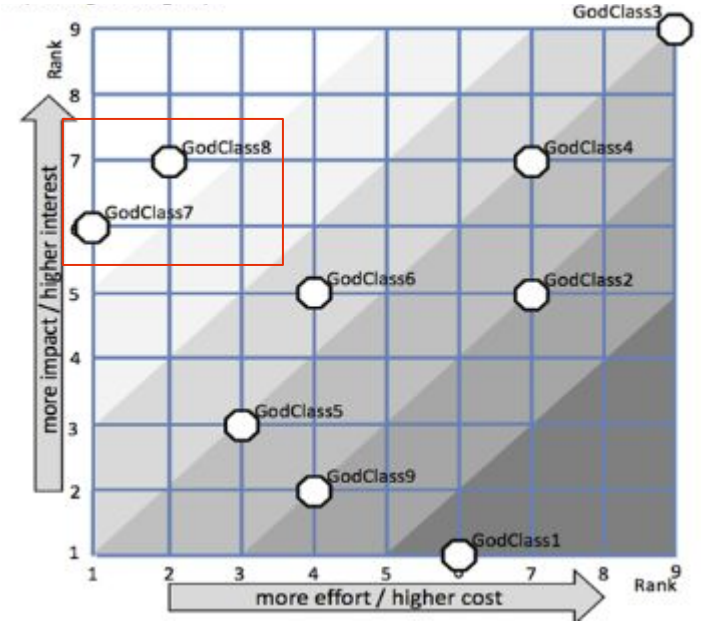
- Auswirkungen von technischen Schulden quantifizieren
  - Wert (Rückzahlungskosten)
  - Zinssatz (potentiellen Kosten durch Aufschiebung)

- Hilfsmittel
  - mathematische Berechnungen
  - definierte Modelle
  - Abschätzung durch Erfahrung und Expertise

Technisches Schuldenmanagement
Priorisierung

- Entscheidungsfindung: Zurückzahlen oder aufschieben
  - Priorisierung von technischen Schulden

- Modelle
  - Kosten-Nutzen-Analyse
  - Portfolio-Ansatz
  - Optionsstrategie
  - Analytic-Hierarchy-Process



Technisches Schuldenmanagement
Prävention

- technische Schulden verhindern / minimieren durch:
  - **Qualitätssicherung**
  - Programmierrichtlinien
  - automatisierte Tests
  - Code-Reviews
  - einheitliche Definition of Done

Technisches Schuldenmanagement
Überwachung

- Grundlage für fundierte Entscheidungen

- Beobachtung von
  - Wert- und Zinsveränderungen über die Zeit
  - Produktivität des Teams
  - Gesamtqualität der Software

Technisches Schuldenmanagement
Dokumentation

- Identifizierte technische Schuld dokumentieren
  - Position, Typ, Beschreibung, …

- zentraler Ort für technische Schulden
  - transparent
  - nachvollziehbar
  - zugänglich

- separates Backlog für technische Schulden
  - Sprint-Planning I

Technisches Schuldenmanagement
Kommunikation

- Entwicklungsteam
  - gemeinsames Verständnis über Prioritäten & Status der Schulden
  - regelmäßige Reviews / Retrospektiven

- Stakeholder
  - Einbeziehung von Rückzahlungen in den Entwicklungsprozess
  - Vermittlung zwischen technischen und nicht-technischen Stakeholdern
  - Implikationen bei Nicht-Rückzahlung verdeutlichen, bspw:
    - Performance beeinflusst
    - Sicherheit gefährdet
    - Entwicklungsaufwand erhöht

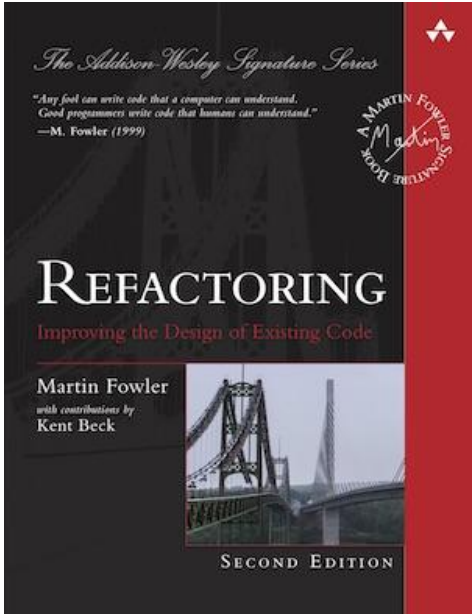
Technisches Schuldenmanagement
Rückzahlung

- Code-Schulden
  - Bug-Fixing
  - Tests

- tieferliegende Schulden
  - Reengineering
  - **Refactoring**

Refactoring
Einführung

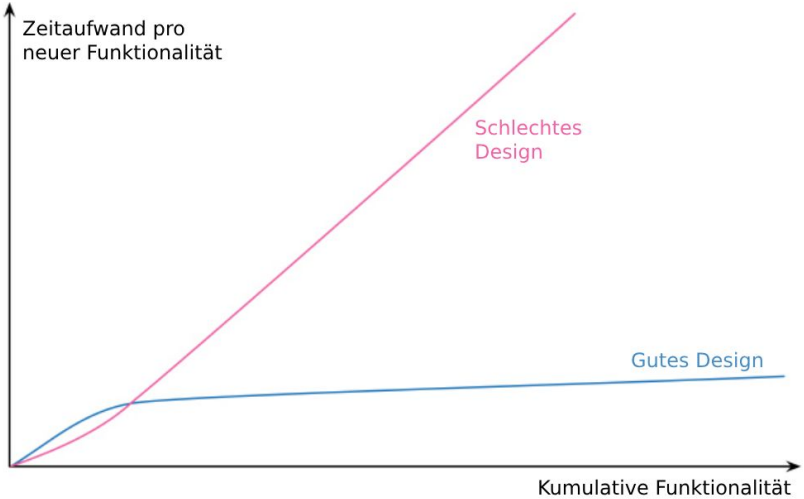
Quelle: https://martinfowler.com/



Quelle: https://refactoring.com/

Refactoring
Motivation

- Architektur zersetzt sich mit Zeit von alleine
- Refactoring verbessert Lesbarkeit des Codes
- Refactoring erleichtert die Fehlersuche
- Verschnellert den Entwicklungsprozess



Quelle: Refactoring: Improving the Design of Existing Code (M. Fowler)

Refactoring
Voraussetzung

- Automatische Tests
- Continuous Integration

Refactoring
Kategorisierung

- Floss Refactoring
  - Wie Zahnseide ein alltäglicher Prozess
  - keine gesonderte Aufgabe
  - opportunistisch
- Root Canal Refactoring
  - Expliziter Auftrag: Refactoring
  - Eigenständige Aufgabe
  - gilt zu vermeiden

Refactoring
Methodik

- Die Beste Zeit ist direkt vor Implementierung
- Wann auch immer ich dadurch die Verständlichkeit des Codes verbessern kann
- Root Canal Refactoring gilt zu vermeiden
    - Große Refactoring-Vorhaben müssen nicht als ganzes gelöst werden
- Als Bestandteil von Code Reviews
- Auch guter Code wird Refactoring benötigen

Refactoring
Anwendungsbeispiel

```
class Student {
    public String StudentName;
}
```

```
class Student {
    private String studentName;

    public String getName() {
        return this.studentName;
    }

    public void setName(String name) {
        this.studentName = name;
    }
}
```

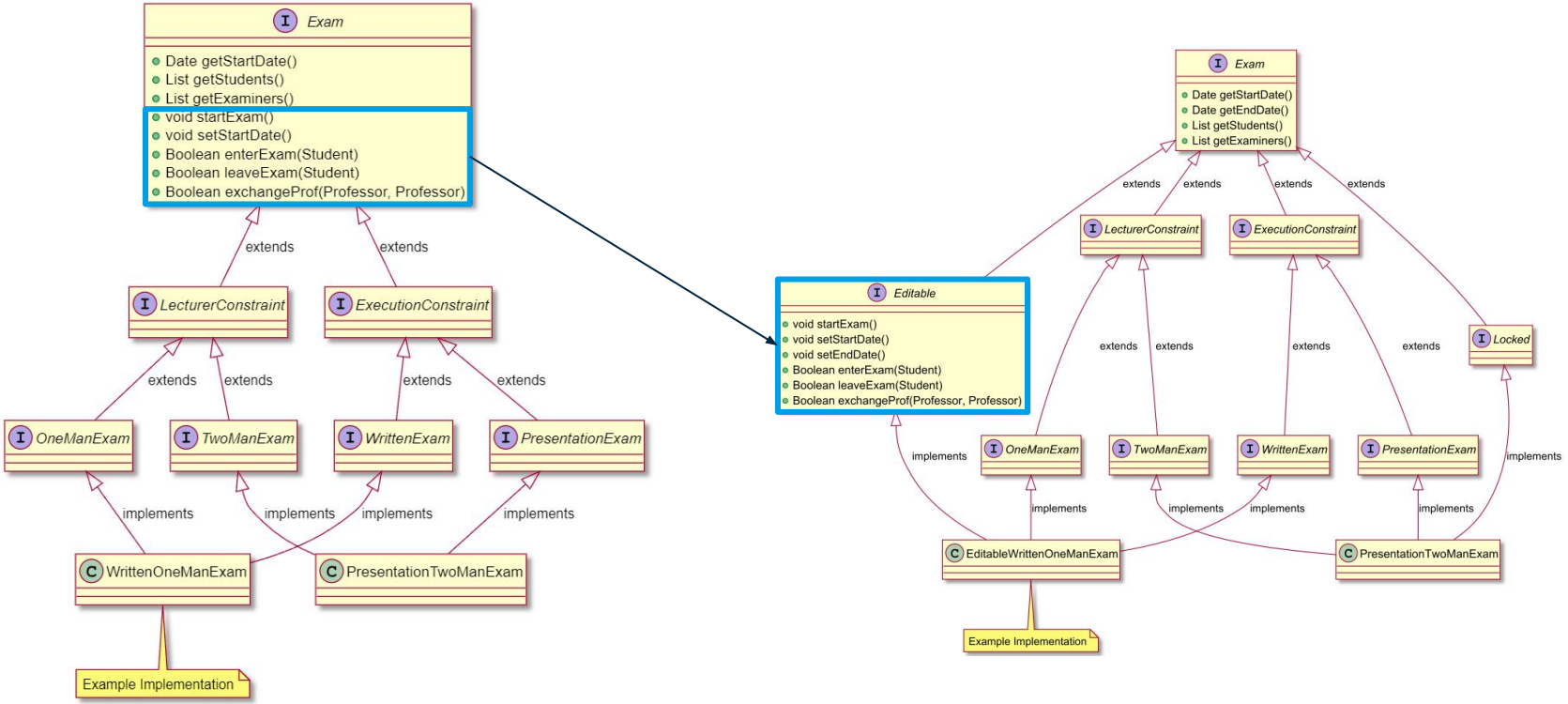
Refactoring
Anwendungsbeispiel

```
if (date.isBefore(plan.summerBreakStart) && date.isAfter(plan.summerBreakEnd)) {
    ticketFee = dayCount * plan.summerBreakFee + plan.regularCharge;
} else {
    ticketFee = dayCount * plan.semesterFee;
}
```



```
if (summerBreak()) {
    ticketFee = summerBreakCharge();
} else {
    ticketFee = regularCharge();
}
```

Refactoring
Anwendungsbeispiel Architektur



Refactoring
Herausforderungen

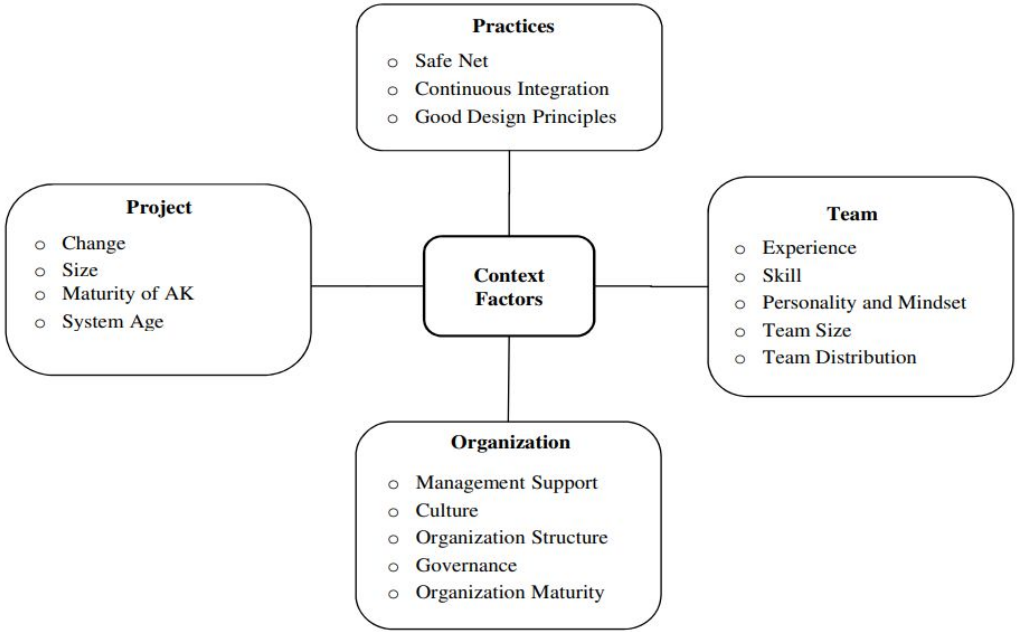
- Gefahren von verschobenen Refactoring
  - Verringerte Produktivität
  - Erhöhte Arbeitslast
  - Höhere Fehleranfälligkeit
  - Später anfallende hohe Arbeitslast durch notwendige Aufräumarbeiten

Refactoring
Herausforderungen

- Kosten müssen immer direkt bezahlt werden
- Konflikte:
  - Zeitlicher Aufwand vs. (sichtbarer) Gewinn
  - Unterschiedliches Verständnis von Aufwandsschätzungen

- **Außerdem:** Refactoring benötigt zwingend Ressourcen zur korrekten Umsetzung
  - Scrum-Phase Planning 2 bietet sich hier besonders an

Refactoring
Erfolgsfaktoren



Quelle: Towards an Evidence-Based Understanding of Emergence of Architecture Through Continuous Refactoring in Agile Software Development

Refactoring
Erfolgsfaktoren

- Wissensvermittlung innerhalb von Teams ist essentiell
- Im Planungsprozess sollten verschiedene Gruppen mit einbezogen werden

Fazit

- keine einheitliche Definition von TS
- häufig fehlende Kenntnisse im Team
- erfolgreiches TSM durch ganzes Team
- Refactoring ist bewährtes Mittel zum Abbau TS
- Grundlage ist immer eine automatische Testabdeckung
- Auch Refactoring muss durch komplettes Team unterstützt werden

Diskussion

- **Erfahrungen mit technischen Schulden?**
  - Sind euch technische Schulden bei der Arbeit begegnet?
  - Wurde der Begriff “Technische Schuld” verwendet?
- **Umgang mit Refactoring?**
  - Gehört Refactoring bei euch zum alltäglichen Entwicklungsprozess?
  - Wie viel Zeit verbringt ihr mit Refactoring?

Quellen

An exploration of technical debt
Edith Tom, Aybüke Aurum, Richard Vidgen
https://www.sciencedirect.com/science/article/pii/S0164121213000022

Experience Report - The WyCash portfolio management system
Ward Cunningham
https://dl.acm.org/doi/10.1145/157710.157715

Managing Technical Debt in Software-Reliant Systems
Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R, Ozkaya, I, Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N.
https://www.researchgate.net/profile/Philippe-Kruchten/publication/221560462_Managing_technical_debt_in_software-reliant_systems/links/0fcfd50b83582ece7d000000/Managing-technical-debt-in-software-reliant-systems.pdf?origin=scientificContributions

How do software development teams manage technical debt? – An empirical study
Jesse Yli-Huumo, Andrey Maglyas, Kari Smolander
https://www.sciencedirect.com/science/article/pii/S016412121630053X

Quellen

Technical Debt Quadrant
Martin Fowler
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Defining, Measuring, and Managing Technical Debt
Ciera Jaspan, Collin Green
https://ieeexplore.ieee.org/document/10109339

A Systematic Mapping Study on Technical Debt and Its Management
Zengyang Li, Paris Avgeriou, Peng Liang
https://www.researchgate.net/publication/269396520_A_Systematic_Mapping_Study_on_Technical_Debt_and_Its_Management

Using technical debt data in decision making: Potential decision approaches
C. Seaman
https://ieeexplore.ieee.org/document/6225999

Quellen

Analyzing the concept of technical debt in the context of agile software development: A systematic literature review
Woubshet Nema Behutiye, Pilar Rodríguez, Markku Oivo, Ayşe Tosun,
https://www.sciencedirect.com/science/article/abs/pii/S0950584916302890

Technical debt and agile software development practices and processes: An industry practitioner survey
Johannes Holvitie, Sherlock A. Licorish, Rodrigo O. Spínola, Sami Hyrynsalmi, Stephen G. MacDonell, Thiago S. Mendes, Jim Buchan, Ville Leppänen
https://www.sciencedirect.com/science/article/pii/S0950584917305098

Managing technical debt: An industrial case study
Z. Codabux and B. Williams
https://ieeexplore.ieee.org/document/6608672

A simulation study of practical methods for technical debt management in agile software development
I. Griffith, H. Taffahi, C. Izurieta and D. Claudio
https://ieeexplore.ieee.org/document/7019961

Quellen

A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team
Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., Succi, G.
https://link.springer.com/chapter/10.1007/978-3-540-85279-7_20

Towards an Evidence-Based Understanding of Emergence of Architecture Through Continuous Refactoring in Agile Software Development
L. Chen and M. A. Babar
https://ieeexplore.ieee.org/document/6827119

Perspectives on refactoring planning and practice: an empirical study
Chen, J., Xiao, J., Wang, Q.
https://link.springer.com/article/10.1007/s10664-015-9390-8

Refactoring: Improving the Design of Existing Code
Fowler, M. and Beck, K. and Brant, J. and Opdyke, W. and Roberts, D.
https://martinfowler.com/books/refactoring.html

Quellen

Identification and management of technical debt: A systematic mapping study
Nicolli S.R. Alves, Thiago S. Mendes, Manoel G. de Mendonça, Rodrigo O. Spínola, Forrest Shull, Carolyn Seaman
https://www.sciencedirect.com/science/article/pii/S0950584915001743

Measure it? Manage it? Ignore it? software practitioners and technical debt
Neil A. Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L. Nord, and Ian Gorton
https://doi.org/10.1145/2786805.2786848