FH Vorarlberg
University of Applied Sciences

Lerntagebuch

Software Lebenszyklus und Qualität

Fachhochschule Vorarlberg

Christina Tschol
Florian Bechtold
Dornbirn, Jänner 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Lerntagebuch ist gegliedert nach den vier stattgefundenen Vorlesungen im November bzw. Dezember. Die in den jeweiligen Vorlesungen präsentierten Folien und Inhalte wurden von uns nach der Vorlesung noch einmal wiederholt und in das Tagebuch abgelegt. Dabei soll erwähnt werden, dass nicht alle Inhalte der Vorlesung in dem Tagebuch beinhaltet sind und somit keine Zusammenfassung der Vorlesung Software Lifecycle und Qualität sein soll. So wurden einzelne, für uns teilweise in der Vorlesung unklare Themen, teils über weitere Recherchen weiter ausgeführt.
Die Hausübung zum Thema der **Quadratischen Gleichung** wird am Ende des Lerntagebuches vorgestellt und erläutert. Erklärungen zu verschiedensten Abkürzungen und Begrifflichkeiten sind im Glossar am Ende des Lerntagebuches zu finden.

2 Vorlesung 1 (28.11.2018)

In der ersten Vorlesung am 28.11.2018 wurden Teile des ersten Foliensatzes **SWE P01 Introduction** und Teile des zweiten Foliensatzes **SWE P02 Motivation** präsentiert.

2.1 Foliensatz SWE P01 Introduction

Im Foliensatz **SWE P01 Introduction** wurden uns organisatorische Inhalte, als auch einen kurzen Überblick über die verfügbare Literatur und die Kursinhalte näher gebracht.
In den Opening Remarks wurde vor allem die Benotung der Studenten besprochen. Dabei wurde entschieden, dass die Erfassung der Leistungen der Studenten über das Tagebuch festgelegt wird. Es wurde ebenfalls ein kurzer Überblick über den Kurs gegeben. Während dem Überblick gab es noch einzelne Begriffserklärungen, welche auch in das Glossar eingeflossen sind.

2.2 Foliensatz SWE P02 Motivation

Im zweiten Teil der Vorlesung wurde der Foliensatz **SWE P02 Motivation** bis zur Folie 30 präsentiert.
Besonders interessant für uns waren dabei die sog. *Selected Events*. Da die meisten Studenten sich einen Bezug zur Realität gewünscht haben, waren für uns die Beispiele in diesem Abschnitt äußerst interessant. Dabei wollen wir auch gleich erwähnen, dass die Erzählungen und persöhnlichen Erfahrungen im Berufsleben von Herrn Prof. Wenzel immer einen sehr positiven Eindruck auf uns gemacht haben und sehr interessant waren.

2.2.1 Allgemeines

Im Abschnitt der *Important Terms and Concepts* haben wir folgendes für uns mitgenommen:

2.2.1.1 Software als Produkt

Produkte können mehr als nur ein physisches Produkt beinhalten: Dienstleistungen, Wasserversorgung, usw.

Sie können somit aus vielen verschiedenen Quellen entstehen. Kurze Anmerkungen zu Produkthaftung (in Bezug zu der Erzählung von Herrn Wenzel mit dem selbst gebautem Tauchgerät)
Grundsätzlich: Produkt wird mit einem Zweck zur Verfügung gestellt.
Zur Software als Produkt gehört nicht nur die Software an sich, sondern auch die Komponenten rund um die Software:

- Dokumentation
- Konfigurationsdateien
- Sonstige Tools

Software wird in verschiedene Produkte kategorisiert (Produktfamilien):

- Generische Software
- Spezifische Software (entwickelt für einen bestimmten Kunden)
- Mischsoftwaren (Softwareprodukte mit hohem Anteil von bestehenden Modulen)

Unterschiede zwischen agiler und traditioneller Softwareentwicklung:

- Agil
  - Schnelles lauffähiges Ergebnis
  - Viel Feedback vom und mit dem Kunden
- Traditoinell
  - Mehr Planung
  - Weniger Rücksprache mit Kunden

2.2.1.2 Software Engineering

Software Engineering beschäftigt sich damit, Probleme mit Software zu lösen - Computer Science hingegen beschäftigt sich mit den Theorien und Methoden hinter Computern und Software. Ein **Software Prozess** ist die Folge von Aktivitäten um ein Softwareprodukt zu entwickeln; dieser beinhaltet folgende Komponenten:

- Software Specification
- Software Development
- Software Validation
- Software Evolution

Diese Komponenten werden von der Qualität und den anfallenden Kosten beeinflusst.
Eine Software Engineering Methode ist ein strukturierter Ansatz, eine Software zu entwickeln. Es geht darum, eine möglichst hohe Qualität mit möglichst wenig Kosten zu ermöglichen. Erfahrungen, Regeln bzgl. Strukturierung eines Codes spielt dabei eine große Rolle.
Wichtiger Begriff: Qualität
ISO9000 – große Norm für Qualitätssicherung: Diese Norm definiert Grundlagen und Begriffe zu Qualitätsmanagementsystemen.
Anforderungen: eine Anforderung ist erst dann vollständig spezifiziert, wenn man weiß wie man die Erfüllung überprüft.

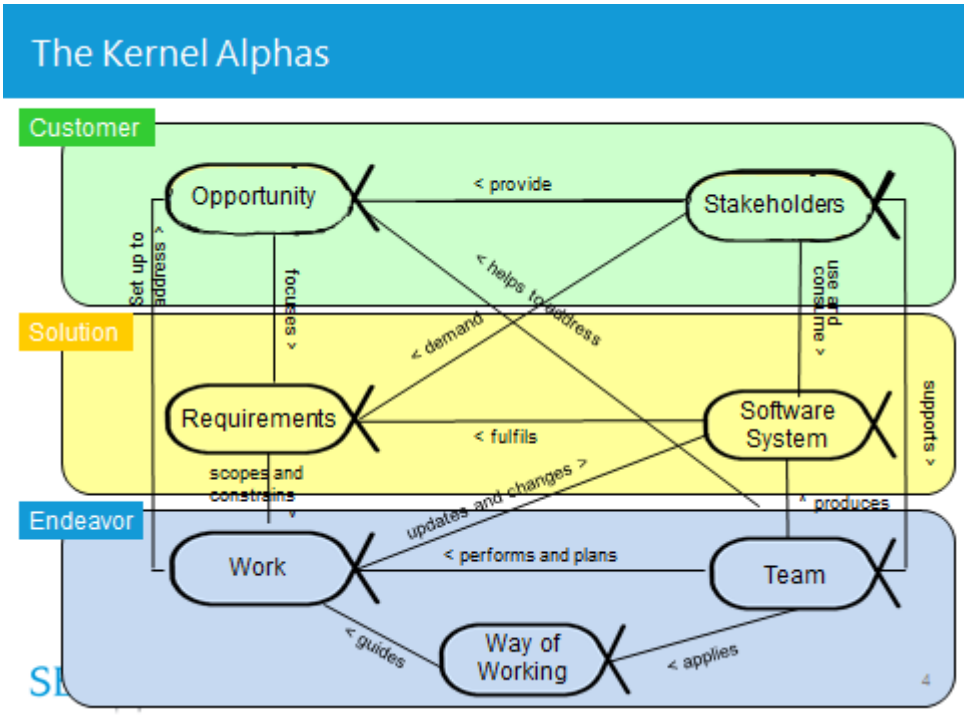
2.2.2 Essence Kernel

Der **Essence Kernel Overview** ist ein Vorschlag, wie man ein Softwareentwicklungsprozess beobachten und einhalten kann. Sehr abstrakter Vorschlag → ganz egal ob nach einem Modell oder Agil gearbeitet wird → kann immer darauf abgebildet werden.
Aufteilung in drei verschiedene Ebenen (oberste Ebene):

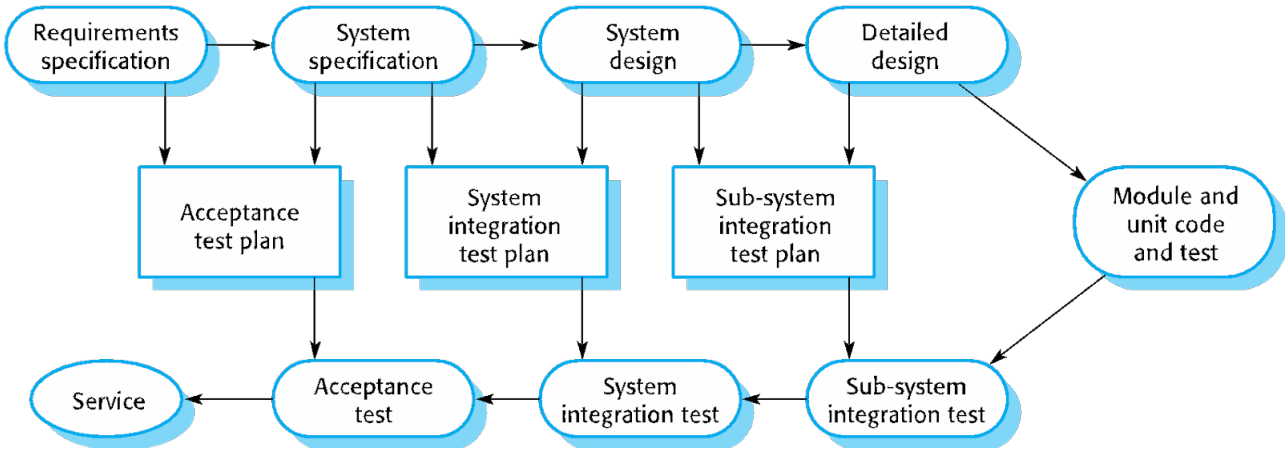
- Customer: was den Anwender interessiert
  - Stakeholders: Interessenten (in irgend einer Form) am Projekt (sind aber nicht Shareholders!); Entwicklungsteam gehört dazu; Stakeholders sorgen auch für das Finanzielle
  - Opportunity: Stakeholders sehen gewisse Chancen im System (Performanz, ...); kann man auch von Zielen sprechen
- Solution: schlussendlich das Ergebnis
  - Requirements: von Ideen der Stakeholder (Opportunities) zu Anforderungen; sind Anforderungen spezifiziert sollten sich Stakeholder noch damit identifizieren und wiederfinden können (ansonsten Übersetzungsproblem)
  - Software System: das Ergebnis

- Endeavor: wo sich Softwareentwickler und Manager aufhalten
  - Team: Leute die arbeiten
  - Work: die Arbeit die zu leisten ist; Work haben die Opportunities im Hinterkopf (Rückkopplung)
  - Way of Working: das Team passt den Way of Working an ihre Aktivitäten an (falls Fehler oder sonstiges festgestellt)

→ 7 Dimensionen, an welchen man ein technisches Entwicklungsprojekt steuern und beurteilen kann.
Für alle Alphas gibt es verschiedene Zustände. Die Zustände müssen nicht immer die genaue Anzahl oder dieselben sein. Einzelrequirements können ebenfalls Zustände haben. Noch wichtig sind Checklisten für jeden dieser Zustände. Kann für alle Alphas gemacht werden. So kann genau aufgezeichnet werden, wo man sich in welchem Zustand befindet → daraus kann man sagen, was man im nächsten Sprint erreichen will.



2.2.3 V-Modell



1. Aus Anforderungen werden bereits einzelne Tests definiert
2. Aus Anforderungen werden Systemspezifikationen festgelegt und weitere Tests geplant
3. Nach der Spezifikation kommt das Design (Architektur); weitere Ausführung des Testplans
4. Detailliertes designing der Module und Subsysteme
5. Softwareentwickler schreiben die Module und machen die Tests
6. Ist der Softwareentwickler fertig, wird auf der Subsystem Ebene begonnen zu integrieren und getestet mit dem Subsystem Plan
7. Systemintegrationstest
8. Abnahme des Kunden
9. Ingebrauchnahme

→ ist das Gegenteil von Agil!

2.2.4 Sonstiges

2.2.4.1 Notizen zur ersten Übung

Wichtig: Assoziativgesetz gilt nicht für Gleichkommazahlen → Rundungsfehler
Wenzel Algorithmus:

(a) Summe der positiven Zahlen angefangen mit der absolut kleinsten Zahl

(b) Summe der negativen Zahlen absteigend

(c) Summieren

2.2.4.2 Systemattribute

Soziotechnische Systeme haben systemspezifische Attribute.
Studierender besteht Prüfung nicht → bezieht sich nur auf den einzelnen Studenten.
Fallen Studenten des ganzen Studengangs durch → Fehler im Studiengang.
Menschen sind involviert → nicht deterministisch.

Funktionale Attribute: Attribute die man sofort überprüfen kann
Nicht Funktionale Attribute: Interaktion zwischen Systemkomponenten (bspw. Performanz, Größe, Stabilität, usw. . .)

2.2.4.3 Environmental Influences

Technische Änderungen in Systemen können Auswirkungen auf die soziotechnischen Systeme haben.
Solche Änderungen können Auswirkungen auf die Power Relationsships (Machtgefüge) in einem Unternehmen haben.

3 Vorlesung 2 (05.12.2018)

In der zweiten Vorlesung am 05.11.2018 wurde der zweite Teil des zweiten Foliensatzes **SWE P02 Introduction** und Teile des dritten Foliensatzes **SWE P03 Software Lifecycles** präsentiert.

3.0.1 Allgemeines

In der Vorlesung am 05.12.2018 haben wir folgendes für uns mitgenommen:

3.0.1.1 Risk Management

Risk Identification → mögliche Quelle (Technologien → (Bsp BlockChain – Skaliert nicht))
Organisation → wie stabil ist die Organisation – muss oder wird umorganisiert werden ?! Schwierig bei laufendem Wechsel
Requirements → Hauptgrund für die agile Arbeit

Dabei muss man sich immer fragen → Wer ist betroffen? Wer geht die Risiken ein? Wenn wir die Risiken kennen und wissen wie man damit umgeht ist das die Grundlage dafür damit umzugehen.

Ziel des Risikomanagement

- Cannot guarantee lack of accidents
- reduces probability of accidents
- reduces cost per accident

3.0.2 Sonstiges

4 Vorlesung 3 (12.12.2018)

5 Vorlesung 4 (19.12.2018)

6 Hausübung

Die Hausübung ist im Foliensatz **SWE P02 Motivation** auf der Folie 46 zu finden.

6.0.1 Requirements

Die Requirements stellten sich wie folgt:

- 3 inputs (a,b,c)
- a ungleich 0
- Plausibilität ($b^2 > 4ac$)
- x1 und x2 real
- Fehlerbehandlung
- Inputprüfung
- Korrekte Wurzel
- Reproduzierbarkeit
- maximal 500 msec pro Aufruf
- Incode-Doku
- Exceptions
- Testbarkeit
- Output Float (Liste)
- Genau 10 Stellen
- Auf die nummerische Genauigkeit achten

6.0.2 Implementierung

Für die Implementierung wurde JAVA verwendet und folgende Klassen implementiert:

- Main
- QuadraticEquation
- AIsZeroException
- ImaginaryNumbersException
- QuadraticEquationTests

6.0.2.1 Main.java

```
import java.util.List;
import java.util.Scanner;

public class Main {


    public static void main(String[] args) {
        //Read Input from console
        Scanner s = new Scanner(System.in);
        System.out.println("Enter a value for a: ");
        double a = Double.parseDouble(s.nextLine());
        System.out.println("Enter a value for b: ");
        double b = Double.parseDouble(s.nextLine());
        System.out.println("Enter a value for c: ");
        double c = Double.parseDouble(s.nextLine());
        s.close();


        //Start the timer
        long start_time = System.nanoTime();

        //Create new QuadraticEquation Object
        QuadraticEquation q = new QuadraticEquation(a, b, c);
        try {
            List<Double> results = q.calc();
            // print results
            for(int i = 0; i < results.size(); i++) {
```

```
                System.out.format("%.10f %n", results.get(i));
            }
            //Thread.sleep(1000);
        } catch (ImaginaryNumbersException e) {
            e.printStackTrace();
        } catch (AIsZeroException e) {
            e.printStackTrace();
//          } catch (InterruptedException e) {
//              e.printStackTrace();
        } finally {
            long end_time = System.nanoTime();
            double difference = (end_time - start_time) / 1e6;
            System.out.println("Duration: (msec) ");
            if (difference < 500) {
                System.out.println(difference);
            } else {
                System.err.println(difference);
            }
        }

    }

}
```

6.0.2.2 QuadraticEquation.java

```
import java.util.LinkedList;
import java.util.List;

public class QuadraticEquation {

    // coefficients of quadratic equation ax^2 + bx + c = 0
    public double a;
    public double b;
    public double c;

    //Contains the results x1 and x2
    public List<Double> results;


    //Constructor
    public QuadraticEquation(double a, double b, double c) {
```

```
        this.a = a;
        this.b = b;
        this.c = c;
        results = new LinkedList<Double>();
    }

    //Calculation method for x1 and x2
    public List<Double> calc() throws ImaginaryNumbersException,
        AIsZeroException {
        if (a == 0) {
            throw new AIsZeroException();
        }

        double s1 = (-b + Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);
        double s2 = (-b - Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);

        if (Double.isNaN(s1) || Double.isNaN(s2)) {
            throw new ImaginaryNumbersException();
        } else {
            results.add(s1);
            results.add(s2);
        }
        return results;
    }
}
```

6.0.2.3 AIsZeroException.java

```
public class AIsZeroException extends Exception {
  public AIsZeroException() {
      super("The Quadratic Equation contains imaginary numbers!");
  }
}
```

6.0.2.4 ImaginaryNumbersException.java

```
public class ImaginaryNumbersException extends Exception {
  public ImaginaryNumbersException() {
      super("The Quadratic Equation contains imaginary numbers!");
  }
}
```

6.0.3 Unit Tests

6.0.3.1 QuadraticEquationTests.java

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import java.util.*;

public class QuadraticEquationTests {

    // tested with
    // http://www.math.com/students/calculators/source/quadratic.htm

    @Test
    public void testCalculation() throws ImaginaryNumbersException,
        AIsZeroException {
        List<Double> results = new LinkedList<Double>();

        // first test a = 1, b = 5, c = 1
        results.add(-0.20871215252208009);
        results.add(-4.7912878474779195);

        QuadraticEquation q = new QuadraticEquation(1, 5, 1);
        assertEquals(results, q.calc());
        results.clear();

        // second test a = 1, b = 2, c = 3
        results.add(-0.45861873485089033);
        results.add(-6.541381265149109);

        QuadraticEquation q1 = new QuadraticEquation(1, 7, 3);
        assertEquals(results, q1.calc());
        results.clear();

    }

    @Test
    public void testException() {
        // third test
        QuadraticEquation q2 = new QuadraticEquation(1, 2, 3);
        assertThrows(ImaginaryNumbersException.class, () -> {
```

```
            q2.calc();
        });

        // fourth test
        QuadraticEquation q3 = new QuadraticEquation(0, 2, 3);
        assertThrows(AIsZeroException.class, () -> {
            q3.calc();
        });
    }

}
```

Glossar

ETW Energietechnik und Energiewirtschaft

SQL Structured Query Language

Bash Bourne-again shell