Software Engineering Hausaufgabe 1

719511_Youran Wang

723866_Yannick Fuchs

Neue Herausforderungen der Softwareentwicklung

1. Nebenläufigkeit

Die Nebenläufigkeit beschreibt die Eigenschaft eines Programms mehrere Berechnung, Anweisungen oder Prozesse zeitgleich laufen zu lassen. Dies kann entweder via Multitasking auf einem Prozessor realisiert oder beispielsweise mit einem Mehrkernprozessor oder einem Rechnerverbund in einem Netzwerk erreicht werden.

Dies ist eine neue Herausforderung, da früher die Hardware sehr teuer war und somit die kommerziellen Rechner zu sowas noch gar nicht allein in der Lage waren. Lediglich Rechnerverbunde hätten sich gelohnt, aber diese wurden damals ebenfalls nur von wenigen Instituten, wenn überhaupt eingesetzt.

2. Interaktive Systeme

Bei interaktiven Systemen wird ein viel größerer Wert auf die Nutzerfreundlichkeit gelegt. Neben der Funktion des Systems soll es sich also auch noch gut anfühlen und möglichst einfach sein dieses zu nutzen.

Dies ist eine neue Herausforderung, da früher ja nur Fachpersonal mit Computern gearbeitet hat und lediglich gefordert wurde, dass ein System funktionieren soll. Die Möglichkeiten der Interaktion waren damals auch nicht so umfassend, wie es heute für uns der Fall ist. Neben der Kommandozeile oder digitalen Fenstern, wo wir Knöpfe einprogrammieren konnten, gibt es heute unter anderem Gesten- und Sprachsteuerung.

3. Eingebettete Systeme

Bei eingebetteten Systemen handelt es sich um Computersysteme innerhalb anderer Geräte. Z.B. die Systeme in den ganzen Homatic-Geräten, wie beispielsweise einem intelligenten Kühlschrank oder Toaster, aber auch die Software innerhalb unserer Autos heutzutage.

Dies ist eine neue Herausforderung, da früher die Hardware sehr teuer war und somit aus finanziellen Gründen nicht überall Computer eingebaut werden konnten. Mit der Zeit wurde die Hardware jedoch günstiger als die Software. Dies sorgte dafür, dass heutzutage mit eingebetteten Systemen gearbeitet werden kann.

4. Cyber-Physical-Systems

Bei dieser Herausforderung handelt es sich um Software, welche mit physikalischen Mechanismen verbunden ist und diese reguliert bzw. ihre Daten verarbeitet. Beispiele dafür sind unter anderem Roboter und industrielle Kontrollsysteme.

Auch hier würde ich als Grund des neuen Aufkommens angeben, dass man dies früher aufgtrund der kombinierten Kosten von Hard- und Software nicht so häufig machen konnte und dieses Feld nun immer beliebter wurde, da die Hardware-Kosten in den vergangenen Jahren sanken.

Maßnahmen zur Qualitätssicherung

1. Konsequente Methodenanwendung im Entwicklungsprozess

Das ist gehört zu Konstruktive Maßnahmen. Software-Engineering-Methoden bieten Anleitungen für die Softwareentwicklung. Es umfasst eine Vielzahl von Aufgaben wie Projektplanung und -schätzung, Analyse der Softwaresystemanforderungen, Datenstruktur, Entwurf der Gesamtsystemstruktur, Entwurf des Algorithmusprozesses, Codierung, Test und Wartung. Die Konsequente der Methodenanwendung gewährleistet einen reibungslosen Übergang zwischen verschiedenen Entwicklungsaktivitäten.

2. Einsatz adäquater Entwicklungswerkzeuge (CASE)

Das ist gehört zu Konstruktive Maßnahmen. CASE steht für "Computer Aided Software Engineering". Mit dem Punkt ist gemeint, dass man Software Produkte zur Unterstützung einer oder mehrerer Entwicklungsschritte nutzt.

Dies trägt zur Qualitätssicherung bei, da Computer deutlich genauer sind als Menschen und somit kleinere Fehler durch z.B. mangelnde Aufmerksamkeit vermieden werden. Somit wird später auch weniger Zeit und Kapital beim Ausbessern dieser Fehler verschwendet.

3. Institutionalisierung der Qualitätssicherung

Das ist gehört zu den Organisatorische Maßnahmen. Eine Institutionalisierung der Qualitätssicherung meint, dass wir die Qualitätssicherung in eine eigene Abteilung auslagern, welche sich rein darum kümmern wird. Die Qualitätssicherung soll institutionalisert werden. Um der Software richtig nach dem Projektwünsch zu erreichen, machen wir eine Qualitätssicherung. Dieser Punkt ist in sofern förderlich, da wir nun nicht mehr nur nebenher die Qualität unserer Produkte sichern, sondern eine Abteilung die ganze Zeit darauf achtet, dass während jeder Phase der Software Entwicklung die Qualität gesichert oder sogar erhöht wird.

4. Statische Programmanalyse

Das ist gehört zu Analytische Maßnahmen. Statische Programmanalyse beteutet, dass Programm ohne Programmslaufen analysiert wird. Die Quellcode wird durch Tokenizer, Synaxanalysis, Datenflussanalysis usw analysiert, um die Integrität, Sicherheit, Verfügbarkeit, Reparierbarkeit und andere Eigenschaft zu prüfen. Es kann die Entwicklere helfen, die Problem zu finden, und die Softwarequalitätssicherung zu verbessern.

5. Dynamische Programmanalyse

Das ist gehört zu Analytische Maßnahmen. Gegenseitig des Statische Programmanalyse, Dynamische Programmanalyse braucht mann das Programm zu laufen. Hier handelt es sich um Software-Tests während der Laufzeit des Programms. Unter anderem werden verschiedene, vorher festgelegte Eingabedaten (sogenannte Testfälle) geprüft um Fehler im Code aufzudecken. Dieser Punkt fördert die Qualität in so fern, dass man möglichst viele Fälle bereits vor Auslieferung des Produktes testen und somit bereits vorher möglichst viele offensichtliche Fehler beheben kann. Wir sehen hierbei nämlich einmal, wie sich das Programm bei bestimmten Eingaben verhält und können bei Bedarf Änderungen im Code vornehmen.

Objektorientierte Programmierung in Java

1. Objektidentität und semantische Gleichheit

Objekte haben eine feste Identität. Haben wir z.B. zwei Beispiel-Objekte B1 und B2, so würde bei der Abfrage 'B1 == B2' false raus kommen, selbst wenn wir sie identisch mit den selben Werten initialisiert hätten. B1 und B2 haben nämlich ihre gänzlich eigenen Identitäten.

```
class Test {

    private String secret;

    public Test(String secret) {
        this.secret = secret;
    }
}
```

Nun haben wir diese Klasse und initialisieren einmal demonstrativ B1 und B2.

```
Test B1 = new Test("test");
Test B2 = new Test("test");

if (B1 == B2) {
   System.out.println(true);
} else {
   System.out.println(false);
}
```

Anschließend haben wir auch direkt einen Test hinzugefügt um auf die semantische Gleichheit einzugehen. Dieser Test würde nun false ergeben und daher würde das gesamte Programm false ausgeben.

2. Vererbung

Die Vererbung bedeutet, dass wir Klassen programmieren können, welche alle Eigenschaften und Methoden einer anderen Klasse "erben", also erhalten. Diese lassen sich später ebenso noch überschreiben, überladen und erweitern.

Im Folgenden werden wir zwei Programme angeben, eines was einfach für sich steht und eines, welches vom ersten Programm alles erbt und noch eine eigene Methode mehr haben wird.

Das erste Programm:

```
class Hund {

    private int age;
    private String name;

    public Hund(int age, String name) {
        this.age = age;
        this.name = name;
```

```
    }
}
```

Da wir nun unseren Hund haben, wird es Zeit für die Klasse, welche von ihm erben wird. Der Schäferhund!

```
class Schaeferhund extends Hund {
    private String colour;

    public Schaeferhund(int age, String name, colour) {
        super(age, name);
        colour = colour;
    }

    public void bellen() {
        System.out.println("Der Schäferhund bellt!");
    }
}
```

3. Schnittstellen

Eine Schnittstelle, in Java auch Interface gemeint, ist eine Art Alternative zur Mehrfachvererbung, da diese ja nicht in Java existiert. Eine Schnittstelle ist eine Datei in welcher wir lediglich die Deklarationen einer oder mehrerer Methoden stehen haben. Diese werden von der Klasse, die sie "erben" möchte direkt implementiert. Anstatt "extends" benutzt man hier jedoch das Schlüsselwort "implements". Eine Schnittstelle kann zum Beispiel so aussehen:

```
interface Test {

    public void doSomething();

    public int calculateSomething();
}
```

4. Klassenmethoden und Klassenattribute

Klassenattribute sind im Grunde einfach nur Variablen, welche zu einer Klasse gehören. Es können zu diesen Variablen ebenfalls Zugriffsmodifikatoren, wie z.B. "public", "protected" oder "private" hinzugefügt werden. Die Klasse selbst kann jedoch immer auf ihre Attribute zugreifen.\ Klassenmethoden folgen dem selben Schema, wobei diese jedoch ebenfalls auch "static" sein können. Dies bedeutet, dass man die Methode ebenfalls nutzen kann, wenn man keine Instanz der Klasse initialisiert. Hier haben wir einmal eine Beispielklasse mit ein paar Attributen und Methoden.

```
class Test {

    private int number;
```

```
    private String word;

    public Test(int number, String word) {
        this.number = number;
        this.word = word;
    }

    public static void testing() {
        System.out.println("Done testing.");
    }

    public int calculating(int a, int b) {
        int c = 0;
        c = a + b;
        return c;
    }
}
```

Die erste Methode ist der Constructor selbst, welcher eine Instanz der Klasse initialisiert, die nächsten beiden Methoden jedoch sind einmal eine statische und eine nicht-statische Methode. Unsere Attribute haben wir dann noch ganz oben, nämlich einmal number und word.

5. Generische Klassen und Methoden

Generische Klassen und Methoden sind Klassen bzw. Methoden, in welchen man einen oder mehrere Datentypen spezifizieren kann, welcher dann einen Platzhalter im Code ersetzt. Somit kann man eine Methode bzw. eine Klasse für mehrere Datentypen schreiben.

```
class DataHandler<P> {
    private P data;

    public void setData(P data) {
        this.data = data;
    }

    public P getData() {
        return this.data;
    }
}
```

Wie wir hier sehen, kann man nun dynamisch die selbe Klasse für mehrere Datentypen nutzen und muss nicht für jeden Datentypen den Code einzeln anpassen.