Software Engineering
Übung 2

Jeremias Kilian
Matrikelnummer: 120996

Elisa Kluge
Matrikelnummer: 119599

Bauhaus-Universität Weimar

24. November 2020

1 Neue Herausforderungen der Softwareentwicklung

1.1 Nebenläufigkeit

Nebenläufigkeit beschreibt die voneinander unabhängige Ausführung zweier Prozesse. Im Gegensatz zur Parallelität ist bei der Nebenläufigkeit die Reihenfolge nicht vorbestimmt, in der die Prozesse ausgeführt werden. Nebenläufigkeit bietet die Vorteile der Ressourcen Ausnutzung (es nutzt mehrere Kerne), Fairness (Bedienung mehrerer Benutzer gleichzeitig) und Zweckmäßigkeit (je Thread kann sequentiell programmiert werden).[2] Die Herausforderung bei Nebenläufigkeit besteht darin, dass zum Beispiel durch Sichtbarkeitsprobleme und dem Zugriff auf gemeinsamen Speicher fehlerhafte Ergebnisse generiert werden. Diese würden bei der Ausführung auf einem Thread nicht auftreten. Auch kann es zu Aktivitätsproblemen kommen, die aus Synchronisationsproblemen entstehen - z.B. wenn es zu Verklemmungen oder Aushungern kommt. (Verklemmung: Threads warten gegenseitig auf Ressourcen der anderen, Aushungern: Ressource wechselt nur innerhalb einer Gruppe von Threads, Threads außerhalb der Gruppe erhalten die Ressource nie[1]) Eine weitere Herausforderung stellt die Performanz dar: es müssen Kontext-Wechsel sowie oben genannte Warteprobleme gelöst werden.[2]

1.2 Interaktive Systeme

Interaktive Systeme zeichnen sich durch vielfältige Wege der Interaktion mit dem Nutzer aus. Beispiele hierfür sind Spracheingabe, Touch-Displays und Eye-Tracking. Somit finden wir interaktive Systeme unter anderem in Smartphones (Apple: Siri). Die Herausforderungen sind hierbei: Auswahl der geeigneten Client-Technologie, Voraussehen von möglichen Anwendungsfehlern, Benutzerfreundlichkeit und Vermeidung des Ausschlusses einzelner Nutzergruppen. Ebenso ist ein hohes Maß an dynamischer Reaktion des Systems erforderlich, da sich die Bedingungen durch die Interaktion ständig ändern.

1.3 Eingebettete Systeme

Eingebettete Systeme schleichen sich immer mehr in unseres alltägliches Leben: bei der Programmauswahl der Waschmaschiene, dem einprogrammierten Rezept im Herd und auch im Auto begegnen sie uns. Sie haben durch die Verbindung von Programmierung und Mechanik eine hohe Komplexität und die Integration der Software in die Mechatronik brigt die Herausforderungen.

1.4 Cyber-Physical-Systems

Cyber-Physical-Systems (CPS) enthalten mechanische Elemente, welche über Netzwerke und IT miteinander verbunden sind. Die physische Umgebung wird dabei erfasst, analysiert und beeinflusst (Sensoren und Aktoren). Somit lassen sich komplexe Systeme und Anlagen, z.B. in der Industrie, kontrollieren und steuern. Ein großes Einsatzfeld ist daher die Industrie 4.0 und das Internet of Things (IoT). Herausforderungen sind unter Anderem eine latenzarme Kommunikation, die Robustheit der Schnittstellen, die Implementierung unterschiedlicher Protokolle (Kopplung von Modulen unterschiedlicher Hersteller) und hohe Anforderungen an die Sicherheit.

2 Maßnahmen zur Qualitätssicherung

2.1 Konsequente Methodenanwendung im Entwicklungsprozess

Damit ist die konsequente Anwendung von Modellierungs- und Softwareengineering Methoden gemeint. Das Anwenden dieser Methoden bildet die Grundstruktur der Software und ist daher für ihre Qualitätssicherung unerlässlich.

2.2 Einsatz adäquater Entwicklungswerkzeuge (CASE)

Bei der Softwareentwicklung sollte das CASE (Computer-aided software engineering) -Prinzip angewendet werden. Dieses besagt, dass der Softwareentwickelnde intensiv Tools nutzt, welche bei Planung, Entwurf

und Dokumentation helfen. Solche Tools sind beispielsweise UML-Diagramme, Strukturierte Analyse und Strukturiertes Design, ROOM, Entity-Relationship-Modelle und weitere.[3] Dies trägt zur Qualitätssicherung bei, da z.B. bei UML und ER Modellen eine gut strukturierte Übersicht aus den schriftlichen Anforderungen gewonnen wird.

2.3 Institutionalisierung der Qualitätssicherung

Es verbessert die Softwareentwicklung, wenn in Unternehmen dedizierte Abteilungen oder Arbeitsgruppen ausschließlich für die Qualitätssicherung zuständig sind. So können Styleguides erstellt sowie Codequalität geprüft und Codetests durchgeführt werden.

2.4 Statische Programmanalyse

Der Sourcecode wird analysiert um mögliche Probleme zu finden. Hierfür ist es wichtig, dass alles entsprechend dokumentiert ist. So kann auch die Entwicklung besser nachvollzogen werden, was zur Sicherung der Qualität beiträgt.

2.5 Dynamische Programmanalyse

Bei der dynamischen Programmanalyse wird zur Laufzeit nach Fehlern oder Problemen gesucht. Die zu testende Software muss somit bereits kompilierbar und ausführbar sein. Dabei werden zum Beispiel Tests (Test Cases) durchgeführt, um die Funktionalität und Performance zu prüfen. Damit wird die Qualität gewährleistet.

3 Eignung von Vorgehensmodellen

3.1 a) Rotationsplanungssystem Krankenhaus

Für die Entwicklung dieser Software eignen sich Prototyporientierte Modelle. Da die Spezifikationen noch nicht klar sind und der Kunde sich zwischen verschiedenen Ansichten noch nicht entschieden hat, ist eine Validierung dieser durch Prototypen für die Benutzerschnittstelle (Krankenhaus) vorteilhaft. Durch Prototypen wird dem Kunden eine Auswahl an Ansichten geboten. All diese Punkte sprechen für das Prototyporientierte Modell. [1]

3.2 b) Spaceshuttle Software

Da dies ein Großprojekt ist, welches im Umfang erweiterbar sein soll, eine praktische Erprobung (Prototyp) erfordert und Anforderungen anpassbar sein sollen, empfiehlt sich für diese Software das Spiral-Modell. Auf die genannten Eigenschaften ist das Spiral-Modell zugeschnitten, da es sich für Großprojekte eignet, Prototypen integriert und wiederholt gleichförmige Phasen des Entwicklungszyklus durchläuft.

3.3 c) Spiel App

Bei diesem Projekt sollte das Extreme Programming verwendet werden, da bei diesem Modell eine enge Zusammenarbeit mit den potentiellen Kunden besteht. Das Modell zeichnet sich durch seine Einfachheit, schnelles Feedback und konsequentes Testen aus.[1]

3.4 d) Medizingerät Software

Für dieses Projekt eignet sich das Vorgehen nach dem Lebenszyklus-Modell, da es ein klar strukturierter Entwicklungsprozess sein soll. Tests stehen dabei im Hintergrund, der Fokus wird auf eine gute Dokumentation gelegt. Beim Lebenszyklus-Modell finden wir eben diese Zwischendokumentation. Wie in der Projektbeschreibung gefordert ist, erhalten wir beim Vorgehen nach diesem Modell ein strukturiertes Softwareprodukt.[1]

Literatur

[1] Prof. Dr. Martin Leucker , Vorlesung *Software Engineering*, 2020/2021.

[2] https://www.hs-augsburg.de/~meixner/prog/html/nebenlaeufigkeit/Nebenlaeufigkeit.html, Zugriff: 15.11.2020 22:50 Uhr

[3] https://de.wikipedia.org/wiki/Computer-aided_software_engineering, Zugriff: 20.11.2020 16:34 Uhr