

Gaida, Lässer, Schwendinger

Lerntagebuch

LV: Softwareprozesse und Softwarequalität

University of Applied Sciences: Vorarlberg

Department of Computer Science
Dipl. Inform. Bernd G. Wenzel

Dornbirn, 2014

Abstract

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Abstract	iii
1 LV 1 am 21.02.2014	1
1.1 Organisatorisches	1
1.2 Motivation des Software-Engineering	1
2 LV 2 am 07.03.2014	3
2.1 Anforderungsanalyse	4
3 LV 3 am 21.03.2014	5
3.1 Anforderungsanalyse	5
3.1.1 Sammeln von Anforderungen	5
3.2 Klassifizierung von Anforderungen	6
3.3 Validierung von Anforderungen	6
3.4 Priorisierung von Anforderungen	6
3.5 Dokumentation und Verwaltung	7
3.5.1 Modelle	7
4 Glossar	9
5 Zusammenfassung	11

Listings

Abbildungsverzeichnis

Todo list

1 LV 1 am 21.02.2014

1.1 Organisatorisches

Zu Beginn dieser Vorlesung wurden zuerst die organisatorischen Randbedingungen für diese Lehrveranstaltung geklärt.

Dabei wurde entschieden, dass anstelle einer Prüfung ein Lerntagebuch in Gruppen zu max. drei Personen erstellt wird. Das Lerntagebuch enthält zusätzlich ein Glossar mit den wichtigsten Begriffsdefinitionen rundum das Thema Softwareprozesse und Softwarequalität.

1.2 Motivation des Software-Engineering

Als Einstieg in das Thema dienten ein paar nicht ganz ernst gemeinte "Gesetze". Diese Gesetze sollten uns verdeutlichen, dass Software-Engineering nicht so trivial und einfach ist, wie es oft den Anschein hat. Wie schon die Folgerung von Kleinbrunner besagt

"Wenn eine Programmieraufgabe leicht aussieht, ist sie schwer."

Es wurden unter anderem die "Gesetze" von Gutterson, Farvour, Brunk, Munbright und vielen anderen vorgestellt.

Um die Bedeutung von qualitativ hochwertiger Software zu verdeutlichen, stellte der Vortragende einige Probleme in bekannten Softwareprojekten wie beispielsweise der Verlust einer Segelyacht mit Crew vor der mexikanischen Pazifikküste aufgrund eines Softwarequalitätsfehlers.

Abschließend erhielten wir die Aufgabe bis zur nächsten Vorlesung ein Programm zu entwickeln zur Berechnung der reellen Lösungen einer quadratischen Gleichung. Ziel bei dieser Aufgabe ist es auf die Qualität des Codes zu achten.

2 LV 2 am 07.03.2014

Zu Beginn wurden kurz die Lösungen der Hausaufgabe besprochen. Es wurde verdeutlicht, dass sogar so eine einfache Aufgabenstellung von einigen Studenten anders implementiert wurde, als ursprünglich gedacht.

Aufbauend auf LV 1 wurden wichtige Begriffe wie Software, Software-Engineering, Softwareprozess usw. definiert. Die wichtigsten Definitionen, auch von zukünftigen Vorträgen, sind im Glossar angeführt. Dabei wurde auch das Vorgehensmodell und die Methode des Software-Engineerings vorgestellt.

Bei der Methode des Software-Engineerings geht es um einen strukturierten Ansatz für die Software-Entwicklung mit Ziel eine möglichst hohe Qualität der Software bei gleichzeitig geringen Entwicklungskosten zu erreichen.

Das Vorgehensmodell definiert wie der Softwareentwicklungsprozess abläuft. Bekannte Modelle sind dabei Scrum, Wasserfall oder das V-Modell. Dabei stellt sich heute oft die Frage, ob man sich für ein klassisches oder agiles Vorgehensmodell entscheidet. Der größte Unterschied zwischen agil und klassisch ist, dass beim Agilen Ansatz der Kunden viel mehr involviert ist und öfters das Feedback von ihm in die Entwicklung mit einfließt. Auch sind die Iterationen beim Agilen Vorgehen meist kürzer.

Anschließend behandelten wir das Thema der soziotechnischen Systeme. Dabei wurde der Begriff genauestens definiert, die wesentlichen Eigenschaften und der Lebenszyklus sowie die Umfeldfaktoren vorgestellt.

Nach den soziotechnischen Systemen behandelten wir das Thema der kritischen Systeme. Dabei geht es um Systeme dessen Ausfall oder Fehlfunktion sich auf die Umgebung in Form von z.B. wirtschaftlichen Verlusten, Schäden an der Umwelt auswirken. Wichtige Faktoren dabei sind die Verfügbarkeit, Zuverlässigkeit, Betriebssicherheit und Systemsicherheit. Der Sammelbegriff für diese Punkte heißt Verlässlichkeit.

Zum Schluss der Lehrveranstaltung wurden die ethischen Herausforderungen beim Software-Engineering behandelt. Dabei geht es um Vertraulichkeit gegenüber dem Arbeitgeber und Kunden, um den Schutz von geistigem Eigentum und den Schutz vor Computermissbrauch. Es gibt auch einen Knigge für das professionelle Verhalten des Software-

Engineering definiert durch ACM/IEEE.

2.1 Anforderungsanalyse

TODO Anforderungsanalyse (stakeholder,..) bis (exklusive) seite 46?

3 LV 3 am 21.03.2014

3.1 Anforderungsanalyse

TODO REMOVE: Start Seite 46

3.1.1 Sammeln von Anforderungen

Um Anforderungen zu sammeln gibt es verschiedene Möglichkeiten:

- Fragebogen
- geschlossene Interviews
- offene Interviews

Sehr gut geeignet ist eine Mischung aus allen drei Varianten. Wichtig ist, dass man die Leute interagieren lässt - sprich man benötigt ein gewisses Maß an Flexibilität und muss sehr gut zuhören können.

Bei der Sammlung von Anforderungen treten einige Probleme auf. Eine sehr große Hürde stellt der Jargon und das implizite Wissen dar. Die Leute gehen davon aus, dass jeder ihren Jargon (mit allen Begrifflichkeiten) kennt und versteht. Das implizite Wissen ist den Leuten meistens nicht bewusst (z.B. aus Gewohnheit) und wird von ihnen zurückgehalten. Beim nicht expliziten Wissen, wissen die Leute, dass sie es haben, und gehen davon aus, dass jeder dieses Wissen hat, da es für sie selbstverständlich ist.

Wichtig ist, dass man nicht sofort mit Interviews anfängt, sondern sich erst in die Domäne einliest. Andernfalls ist das Ergebnis von den Interviews nicht zu gebrauchen.

Szenarien (Seite 47) Es ist zu empfehlen bei der Erstellung von Szenarien Ad-hoc vorzugehen und zur Dokumentation UML-Diagramme verwenden. Es kann schon sehr früh mit der detaillierten Dokumentation begonnen werden. Details sind wichtig um alles korrekt verstehen zu können.

Ethnografische Methoden (S. 48) Durch diese Variante kann man durch Beobachten der

Leute Anforderungen herausgefunden. Es müssen stets stillschweigend Notizen gemacht werden und am Ende wird mit einem Experten das Beobachtete diskutiert. Dies sollte man mit der jeweiligen beobachteten Person machen und nicht mit deren Chef, da sonst weitere Beobachtungen durch wenig Kooperation der Mitarbeiter behindert werden. Durch diese Variante kann man sehr gut implizite, nicht explizite Anforderungen und Abweichungen erkannt werden. Auch die Anforderungen der Benutzer können sehr gut ermittelt werden.

3.2 Klassifizierung von Anforderungen

???

3.3 Validierung von Anforderungen

Es gilt zu prüfen, ob die Anforderungen gültig (filtern von falschen und unnötigen Anforderungen), konsistent, vollständig (sehr schwer zu prüfen), realisierbar (genügt das Budget, Zeit? ist die Technologie geeignet?) und verifizierbar sind.

Um dies Überprüfung durchzuführen kann man ein Anforderungsreview machen, einen Prototypen entwickeln oder Testfälle erzeugen.

Für ein Anforderungsreview benötigen wir aus allen wichtigen Gruppen der Stakeholder einen Vertreter, einen Systemarchitekten und einen Vertreter der Softwareentwickler (müssen am Ende ja wissen, was sie entwickeln sollen). Die Durchführung wird von den jeweiligen Anwendervertretern durchgeführt und alle Anforderungen diskutiert. Durch das Anforderungsreview können Konflikte, Widersprüche, Fehler und Versäumnisse ermittelt werden. Sie sollten in einem Review-Bericht festgehalten werden.

3.4 Priorisierung von Anforderungen

Die Priorisierung ist speziell bei einer agilen Vorgehensweise sehr wichtig.

Der Grundgedanke ist durch Phaseweise Implementierung das Leben der Anwender durch kontinuierliche Softwareupdates zu erleichtern. (?????). Es minimiert auch das Risiko der Einführung der Software.

Die Prioritätsfestlegung erfolgt über eine Bewertungsformel. Es wird jeweils eine Bewertung zu jeder einzelnen Anforderung durch das Nutzwertanalyseteam und das Kostenanalyseteam. Beide Teams bewerten völlig unabhängig voneinander und ohne Kommunikation zwischen den Teams. Nach der Auswertung erhält man eine Liste der Anforderungen für die nächste Ausbaustufe.

3.5 Dokumentation und Verwaltung

3.5.1 Modelle

Bieten eine detaillierte und formalisierte Dokumentation über des Ist-Zustands und Soll-Zustands. Es werden auch Sichten von Kunden oder anderen Abteilungen auf beide Zustände festgehalten.

Kontextmodelle

Mit diesem Modell werden die Systemgrenzen hinsichtlich des Gesamtsystems definiert. Hier ist es wichtig den Kontext zu definieren. Neben dem Gesamtsystem werden die technischen

TODO ZU LANGSA;!! SEITE 64

Scope: was man wirklich implementieren will Kontext: das ganze um den Scope herum.

Verhaltensmodelle

Es wird daten- und ereignis-orientiert die Abläufe im System definiert. Es gibt auch Mischformen (z.B. eine Banküberweisung). Zur Veranschaulichung dienen Diagramme wie Datenflussdiagramme, Prozessdiagramme, Zustandsdiagramme, Sequenzdiagramme oder Szenarios.

Datenmodelle

Für die Darstellung der Datenmodelle gibt es ER-Diagramme, UML-Klassendiagramme, XML-Schema- OWL und weitere. Sie dienen zur logischen definition der Datenstrukturen hinsichtlich transient, persistent...

TODO Seite 76.

Ein wichtiges Werkzeug ist das Data Dictionary. Hier speichert man sich die Klassen und Attribute... Man speichert sich zu den Entitäten gewisse Eigenschaften wie zum Beispiel den Namen, Beschreibung und Typ.

TODO Seite 77.

4 Glossar

- Software
- Software-Engineering
- Informatik
- Softwareprozess
- Software-Validierung (o Ob die Software für den Kunden einen Wert hat)
- Software-Verifizierung (o Übereinstimmung der Software mit den Spezifikationen)
- Qualität
- Technisches computer-basiertes System
- Soziotechnisches System
- Funktionale Anforderungen
- Nicht-Funktionale Anforderungen
- Technical Debt
- Verifikation
- Validierung
- ESSENCE Kernel
- Data Dictionary

5 Zusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.