UNIVERSITÄT LEIPZIG

LPZ E-BUSINESS applied telematics

Ausgewählte Kapitel der Software-Technologie: Spezifikation und Testen

WS 2006/07

Lehrstuhl für Angewandte Telematik / e-Business
Fachbereich Mathematik und Informatik der Universität Leipzig
Volker.Gruhn@informatik.uni-leipzig.de
0341 / 97 323 31

UNIVERSITÄT LEIPZIG

LPZ E-BUSINESS applied telematics

Agenda

1 Zielsetzung der Vorlesung
- 1.1 Ziele
- 1.2 Motivation
  - Begriffsbildung
  - Relevanz
  - Defizite
  - Einordnung in die Informatik
  - Phänomene des Software Engineerings
  - Software-Krise

2 Software Engineering (SWE)
- 2.1 25 Jahre Software Engineering
- 2.2 Software Engineering - wann braucht man‘s?
- 2.3 Eigenschaften von Software
- 2.4 Prinzipien des SWE

1.1 Ziele

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Kenntnis der grundsätzlichen verschiedenen Dimensionen des Software Engineerings (technisch, organisatorisch, psychologisch)
- Vermittlung eines Überblicks über das Spektrum gängiger Konzepte zur Spezifikation und zum Testen
- Erläuterung ausgewählter Methoden und Sprachen zur Spezifikation
- Einbettung dieser Spezifikationskonzepte in die Softwaretechnik
- Erläuterung verschiedener Testtechniken
- Einbettung in den gesamten Softwareprozess

1.2 Motivation

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Begriffsbildung Software Engineering
  - *The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. [NR68]*
  - *Software Engineering ist die Anwendung wissenschaftlicher Erkenntnisse mit dem Ziel, Computer mittels Programmen, Verfahren und zugehörigen Dokumenten dem Menschen nutzbar zu machen [Den91]*
  - *... [is] the systematic approach to the development, operation, maintenance, and retirement of software." [IEEE83]*

NR68, P. Naur, B. Randell (eds), Software Engineering: Report on a Conference, NATO Scientific Affairs Division, Brüssel, 1968

Den91, E. Denert, Software-Engineering, Berlin, Springer, 1991

IEE83, IEEE Standard Glossar of Software Engineering terminology - IEEE Standard 729, IEEE Computer Society Press, New York, 1983

1.2 Motivation

UNIVERSITÄT LEIPZIG

LPZ E-BUSINESS applied telematics

- Begriffsbildung Software Engineering ff.
  - *A software engineer must of course be a good programmer, be well-versed in data structures and algorithms, and be fluent in one or more programming languages. ... The software engineer must be familiar with several design approaches, be able to translate vague requirements and desires into precise specifications, and be able to converse with the user of a system in terms of the application rather than in ‚computerese'. [GJM91]*
- Daraus folgende, notwendige Fähigkeiten:
  - Kommunikation auf verschiedenen Abstraktionsebenen
  - Erstellung und Verwendung von Modellen / Spezifikationen
  - Kommunikation mit Personen mit unterschiedlichen Zielsetzungen, Vorstellungen, Ausbildungen
  - Arbeitsplanung und -koordination

1.2 Motivation

UNIVERSITÄT LEIPZIG

LPZ E-BUSINESS applied telematics

- Defizite des Software Engineerings
  - Komplexitätsbeherrschung als das Problem
  - Überbewertung des Formalisierbaren
  - Unterbewertung der Anschauung
  - Unterschätzung verfahrensorientierter Arbeitsteilung (bei gleichzeitiger Überschätzung der formalisierbaren Arbeitsteilung)
  - Ausbildungsdefizite

1.2 Motivation

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Phänomen der Software-Entwicklung
  - Trotz der Abhängigkeit von Software gibt es
    - Keine zuverlässige Herstellung von Software im industriellen Maßstab
    - Kosten- und Terminüberschreitungen
    - Bei Auslieferung: ungenügende Softwarereife
    - Keine Produktivitätskontrolle wie in anderen industriellen Fertigungsbereichen
    - Keine Qualitätskontrolle wie in anderen industriellen Fertigungsbereichen, gerade das Testen ist immer noch unterbewertet.

1.2 Motivation

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Ursachen der Phänomene (nach [CKI88])
  - Thin Spread of Application Domain Knowledge
    - Projekte in neuer Branche
    - Technologiezentriertheit
  - Fluctuating and Conflicting Requirements
    - Markt
    - verschiedene Kunden
    - Erkenntnisprozesse
    - Missverständnisse
  - Communication and Coordination Breakdowns
    - unterschiedliche Vorstellungen und Zielsetzungen
    - Inkongruenzen zwischen Kompetenz und Verantwortung
    - Kapitulation

CKI88, B. Curtis, H. Krasner, N. Iscoe, A Field Study of the Software Design Process for Large Systems, Communications of the ACM, November 1988, Vol. 31, No. 11, pp 1268 - 1287

1.2 Motivation

- Phänomen I: Abbruchrate

Anzahl Function Points	Früher als geplant	Termingerecht	Verspätet	Abgebrochen
1 FP	14,86 %	83,16 %	1,92 %	0,25 %
10 FP	11,08 %	81,25 %	5,67 %	2,00 %
100 FP	6,06 %	74,77 %	11,83 %	7,33 %
1.000 FP	1,24 %	60,76 %	17,67 %	20,33 %
10.000 FP	0,14 %	28,03 %	23,83 %	48,00 %
100.000 FP	0,00 %	13,67 %	21,33 %	65,00 %
Durchschnitt	5,53 %	56,94 %	13,71 %	23,82 %

Tabelle 1-1: Abbruchrate großer Projekte nach [Jon96]

Jon96, C. Jones, Large Software System Failures and Successes, in: American Programmer, Vol. 9, No. 5, September 1996

Agenda

2 Software Engineering (SWE)

2.1 25 Jahre Software Engineering

2.2 Software Engineering - wann braucht man's?

2.3 Eigenschaften von Software

2.4 Prinzipien des SWE

2.1 25 Jahre Software Engineering

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- 1968: **NATO-Konferenz in Garmisch-Partenkirchen**
  - Identifikation des Sachverhalts des Anwendungsstaus und Prägung des Begriffs der Software-Krise (nötige Software kann mit den zur Verfügung stehenden Ressourcen nicht entwickelt werden).
- Seitdem**:**
  - Vielzahl technologischer Lösungsansätze, die einzelne Aktivitäten der Software-Entwicklung unterstützen
  - lange Innovationszyklen
  - wenig Ansätze mit industriellen Auswirkungen
  - enorme Produktivitätsverbesserungen, aber keine Produktivitätssprünge
  - zusätzliche Software-Lösungen geraten in Reichweite, der Anwendungsstau bleibt bestehen

2.1 25 Jahre Software Engineering

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Software-Krise
  - **60er Jahre: Spezialrechner mit Spezialsoftware**
    - niedrige Rechenleistung
    - kleiner Hauptspeicher
    - hohe Kosten / punktuelle Rentabilität
    - batchorienterte Software
    - Programme werden meist von den Anwendern erstellt
    - Software-Entwickler sind Autodidakten
    - Programmier*kunst*

2.1 25 Jahre Software Engineering

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Software-Krise ff.
  - **70er Jahre: Mikroelektronik**
    - explodierende Rechenleistung
    - größerer Hauptspeicher
    - höhere Rentabilität
    - Menge der prinzipiell lösbaren Software wächst
    - Software kann nicht mehr von einzelnen erstellt werden

2.1 25 Jahre Software Engineering

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Software-Krise ff.
  - **80er Jahre: Software-Massenmarkt**
    - Trennung von Anwendern und Entwicklern
    - Software-Einsatz orientiert sich am wirtschaftlichen Nutzen
    - Software-Beschreibung bestimmt Vertragsverhältnisse
  - **90er Jahre: Komponenten-Markt und Migration**
    - Integration und Interoperabilität
    - Branchenmärkte
  - **00er Jahre: Konzentration auf Anwendungswissen**
    - Modellgetriebene Entwicklung
    - „Business Aligned“ IT

2.1 25 Jahre Software Engineering

- Die Software-Krise als Katalysator für das Software Engineering
  - Nötig:
    - von der Kunst zur Ingenieurskunst
    - von der individuellen Bastelei zum Produktionsprozess
  - Aber:
    - die Änderungen in den Rahmenbedingungen haben sich nur langsam und unter Schmerzen auf das Software Engineering niedergeschlagen
    - Noch heute wird Software-Entwicklung teilweise als künstlerischer Prozess verstanden und als solcher akzeptiert

2.2 Software Engineering - wann braucht man's ?

- **Software Engineering hat was mit “viel” zu tun:**
  - Die zu erstellende Software hat VIELE Komponenten
  - Die zu erstellende Software wird von VIELEN Anwendern benutzt werden
  - Die zu erstellende Software wird von VIELEN Entwicklern erstellt
  - Die zu erstellende Software wird möglicherweise in VIELERLEI Hinsicht erweitert
  - Die zu erstellende Software soll auf VIELEN Plattformen laufen
  - In der Entwicklung der zu erstellenden Software kommen VIELE Rollen vor

2.2 Software Engineering - wann braucht man's ?

- **Durch Software Engineering sicherzustellende Eigenschaften von Software (Übersicht):**
  - Korrektheit / Zuverlässigkeit / Robustheit
  - Performanz
  - Benutzungsfreundlichkeit (Usability)
  - Wartbarkeit
  - Wiederverwendbarkeit
  - Portierbarkeit
  - Interoperabilität

2.3 Eigenschaften von Software

- Korrektheit:
  - Übereinstimmung eines Programms mit seiner Spezifikation
    - ohne Spezifikation ist die Korrektheitsfrage nicht entscheidbar
    - bei informaler Spezifikation ist die Korrektheitsfrage nicht eindeutig entscheidbar
- Zuverlässigkeit:
  - dauerhafte Einsetzbarkeit, der Anwender kann sich erlauben, von der Software abzuhängen
    - Zuverlässigkeit ist ein relatives Kriterium, das vom Einsatzzweck der Software und vom Schadenspotential der Nichtverfügbarkeit der Software abhängt.
    - Zuverlässigkeit von Software wird bei neuer Software in aller Regel nicht erwartet (Bananen-Software). Wesentlicher Unterschied zu fast allen anderen industriellen Produkten.

2.3 Eigenschaften von Software

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Zusammenhang zwischen Korrektheit und Zuverlässigkeit

	zuverlässig	**nicht zuverlässig**
korrekt	alle Anforderungen spezifiziert und erfüllt	unspezifizierte Anforderungen
nicht korrekt	nicht korrekte Fälle treten nicht auf (Überspezifikation)	Fehler verursachen Fehlverhalten

2.3 Eigenschaften von Software

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Robustheit:
  - Toleranz gegenüber nicht spezifizierter Bedienung / nicht spezifizierten Rahmenbedingungen
    - auch nicht robuste Software kann korrekt sein
    - nicht robuste Software kann leicht nutzlos werden
    - wesentliche Robustheitsanforderungen sollten deshalb spezifiziert werden

2.3 Eigenschaften von Software

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Performanz:
  - Erfüllung der Anforderungen an Antwortzeitverhalten, Ressourcenbedarf
  - prinzipielle Überprüfungsmethoden:
    - Messen
    - Berechnen
    - Simulieren
  - häufiger Umgang mit der Eigenschaft Performanz:
    - Erstellen einer initialen Version
    - Verbesserung zum Zweck des Erreichens der notwendigen Performanz
    - Problem: deutliche Verbesserungen erfordern zuweilen grundlegendes Redesign

2.3 Eigenschaften von Software

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Benutzungsfreundlichkeit (Usability)
  - Software ist benutzungsfreundlich, wenn die Anwender sie für einfach benutzbar halten
    - Gestaltung der Dialoge
    - einfache Konfigurierbarkeit
    - einleuchtender Aufbau (so weit sichtbar)
- alles andere als das Empfinden der Anwender zählt nicht!
- Häufiger Ansatz: Standardisierung der Benutzungsoberflächen

2.3 Eigenschaften von Software

- Definition Usability
  - Usability eines Produktes ist das Ausmaß, in dem es von einem bestimmten Benutzer verwendet werden kann, um bestimmte Ziele in einem bestimmten Kontext effektiv, effizient und zufriedenstellend zu erreichen. (ISO 9244-11)
- Testbare Attribute finden, um Anforderungen an Software abzuleiten
  - Dekomposition der Definitionen der am häufigsten zitierten Autoren
  - Zeilenweise sortiert ähnliche testbare Attribute

Constantine	**Hix**	**ISO 9126**	**Nielsen**	**Preece**	**Shackel**	**Shneiderman**	**Wixon**
Learnability	Learnability	Learnability	Learnability	Learnability	Learnability	Learnability	Learnability
Efficiency in use	Long-term performance	Operability	Efficiency of use	Throughput	Effectiveness	Speed of performance	Efficiency
Remember-ability	Retainability	-	Memorability	-	Learnability	Retention over time	Memorabilit
Reliability in use	-	Operability	Errors	Throughput	Effectiveness	Rate of errors by users	Error rates
User satisfaction	Long-term user satisfaction	Attractiveness	Satisfaction	Attitude	Attitude	Subjective satisfaction	Satisfaction

2.3 Eigenschaften von Software

- Erlernbarkeit (Learnability)
  - wie schnell, wie einfach produktive Arbeit mit neuem System & wie einfach Erinnerung an Art und Weise der Nutzung
- Effizienz (Efficiency of use)
  - Anzahl der Aufgaben/Zeiteinheit, die ein Nutzer ausführen kann
  - Beinhaltet Effektivität (akkurat und komplett Ziele erreichen) und Effizienz (durch auf dem Weg zum Ziel verbrauchte Ressourcen) aus ISO Definition
- Reliabilität, Verlässlichkeit (Reliability in use)
  - Fehlerrate (Nutzerfehler, nicht Systemfehler) bei der Benutzung & Dauer bis zur Wiederherstellung nach einem Fehler
- Zufriedenheit (Satisfaction)
  - subjektive Meinung der Systemnutzer, Komfort und Akzeptanz

➢ Ableitbar davon sind Anforderungen, die in GUI und System umgesetzt werden

2.3 Eigenschaften von Software

Anforderungen an Usability

Zugänglichkeit
- Behinderte unterstützen, Multi-Channeling, Internationalisierung
- + Erlernbarkeit, + Zufriedenheit
- css für printer layout, web layout

Anpassbarkeit
- Anpassung an Nutzer (Erfahrung, Einstellungen, System Erinnerungsvermögen)
- + Effizienz, + Zufriedenheit
- z.B. Skins für Mozilla

Konsistenz
- visuelle, funktionelle, evolutionäre Konsistenz
- + Erlernbarkeit, + Verlässlichkeit
- z.B. typische Shortcuts

Fehlermanagement
- Fehlervermeidung und Fehlerbehandlung
- + Effizienz, + Verlässlichkeit
- z.B. rot unterstrichende Syntaxfehler in Eclipse

Explizite Nutzerkontrolle
- direkte Manipulation erlauben, Kontrolle geben
- + Zufriedenheit
- z.B. Cancel von großen Downloads

Anleitung
- informative, einfach nutzbare, relevante Anleitung in Applikation & Manual
- + Erlernbarkeit, - Effizienz, + Verlässlichkeit
- z.B. To-Do-List ArgoUML

Minimiere kognitive Last
- menschliche Grenzen, max. 7 Items auf einer Seite
- +/- Effizienz, + Erlernbarkeit
- z.B. "auto hide" - Menüs

Natürliche Bilder
- Vorhersagbarkeit, klare Symbolsprache, einfache Navigation
- + Erlernbarkeit, + Effizienz, + Verlässlichkeit
- z.B. Mülleimer: drag'n drop

Feedback
- Systemstatus
- + Effizienz, + Erlernbarkeit
- z.B. Fortschrittsanzeige beim Download

Folmer.E. et al.: *A framework for capturing the relationship between usability and software architecture*

2.3 Eigenschaften von Software

- Wartbarkeit:
  - leichte Handhabbarkeit einer Software nach ihrer Auslieferung
- Arten der Wartung:
  - korrektive Wartung: Beseitigung von Fehlern
  - adaptive Wartung: nachträgliche Anpassung an neue Anforderungen, z.B. gesetzliche Änderungen, neue Organisation
  - perfektive Wartung: Verbesserung im Hinblick auf nicht-funktionale Anforderungen, z.B. Performanz, Ergonomie
- Wartbarkeit hängt stark von Strukturierung ab
- Wartbarkeit nimmt zumeist mit der Lebensdauer der Software ab
  - Lehmans Law of Increasing Software Entropy: die Struktur der Software nimmt mit zunehmender Lebensdauer ab.

2.3 Eigenschaften von Software

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Wiederverwendbarkeit:
  - Wahrscheinlichkeit, mit der Software in einem anderen Kontext wiederverwendet werden kann (oft bezogen auf Komponenten)
- Wiederverwendbarkeit entsteht nicht zufällig, sondern muss geplant sein (infrastrukturell unterstützt werden (z.B. durch Rolle „Wiederverwender“))
- Beispiele: parametrisierte Datenstrukturen, Klassenbibliotheken

2.3 Eigenschaften von Software

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Portierbarkeit
  - die Portierbarkeit einer Software ergibt sich aus dem Aufwand, der nötig ist, um eine Software auf einer anderen Plattform (DBMS, BS, UIMS) lauffähig zu machen (im Verhältnis zu ihrem Entwicklungsaufwand)
- hoher Grad an Portierbarkeit durch Verkapselung von Plattformabhängigkeiten, vgl. Ansätze der modellgetriebenen Softwareentwicklung

2.3 Eigenschaften von Software

- Interoperabilität:
  - eine Software ist interoperabel, wenn sie sich mit geringem Aufwand mit anderer Software integrieren lässt
- PC-Werkzeuge sind mittlerweile großenteils interoperabel
- hochintegrierte Software-Systeme sind meist weniger interoperabel, weil sie „alles“ können
  - Beispiel: integrierte Entwicklungsumgebungen, Workflow-Management-Systeme und Datenmodellierung
- Interoperabilität wird immer mehr zu einem Muss, weil kaum noch Software in völlig neuen Gebieten eingesetzt wird
  - Beispiel: Interoperabilität mit Office-Werkzeugen, Großrechner-Software, Datenmodellierungswerkzeugen

2.4 Prinzipien des SWE

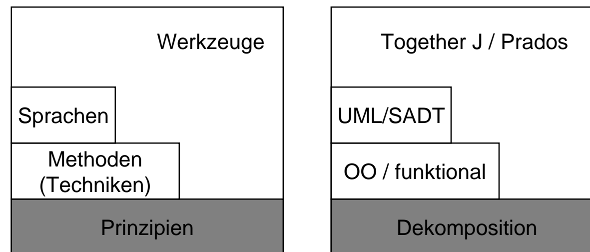
- Überblick (nach GJM91)
  - Striktheit und Formalität
  - Strukturierung
  - Modularität
  - Abstraktion
  - Änderbarkeit
  - Allgemeinheit
  - Inkrementalität

[GJM91]: C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, Prentice-Hall, 1991)

2.4 Prinzipien des SWE

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

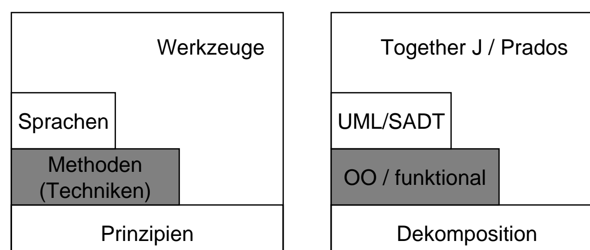
- Prinzip
  - Grundsatz, den man seinem Handeln zugrundelegt



2.4 Prinzipien des SWE

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

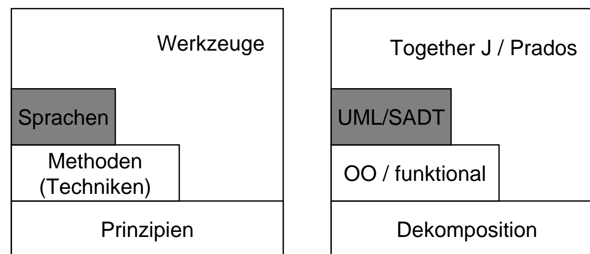
- Technik
  - Vorschrift zur Durchführung einer Tätigkeit (Was ist wie zu tun?)
- Methode
  - Planmäßig anwendbare, begründete Technik zur Erreichung vorgegebener Ziele (Was ist wie und unter welchen Rahmenbedingungen zu tun, so dass ein gutes Ergebnis erreicht wird?)



2.4 Prinzipien des SWE

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Sprache
  - syntaktische Regeln zur Unterstützung einer Methode oder Technik. Eine Sprache besteht aus ihrer Syntax und ihrer Semantik:
    - formale Sprachen haben eine formale Syntax- und Semantikdefinition (Aussagenlogik, Prädikatenlogik)
    - semiformale Sprachen haben eine formale Syntax, aber keine klar definierte Semantik (SADT, UML)
    - informale Sprachen haben weder eine formale Syntax noch eine formale Semantik (Deutsch, Englisch)



2.4 Prinzipien des SWE

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS applied telematics

- Werkzeuge
  - Rechnerunterstützung für Techniken und Methoden