Vorlesung "Softwaretechnik"

Zusammenfassung

Lutz Prechelt

Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Wiederholung einiger Aussagen der 1. Vorlesung
  - Worauf kommt es an?
- Schnelldurchgang durch Stoffplan
  - Alles noch mal Revue passieren lassen
- Worüber wir nicht gesprochen haben
  - Wichtige Themen, die keinen Platz gefunden haben
- Warme Worte zum Abschluss
  - Empfehlungen

Zusammengefasst: Aufgaben der SWT

- Helfen zu klären **OB** (Kosten/Nutzen-Analyse)
  - Was könnte SW hier leisten? Und welchen Nutzen brächte das?
  - Was würde sie kosten?
- Klären **WAS** (Anforderungen)
  - Was sollte die SW tun? (funktionale Anforderungen)
  - Welche Eigenschaften sollte sie mitbringen? (nichtfunktionale A.)
- Klären **WIE** (Entwurf)
  - Wie sollte die SW aufgebaut sein?
- **Bauen** (Implementierung)
- **Validieren** der Lösung (Qualitätssicherung)
  - Hat die SW die erwünschten Eigenschaften?
- **Verwalten** des Vorgehens (Management)
  - Den ganzen Prozess, mit dem obige Aufgaben gelöst werden, gestalten und verwalten

Alle 6 Aufgaben sind eng miteinander verwoben

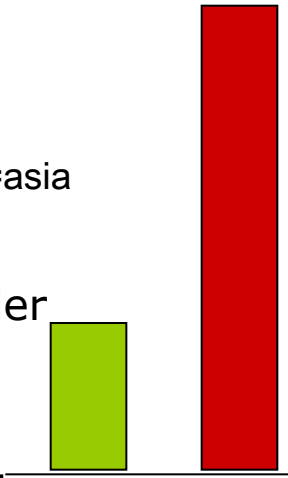
DIE NACHRICHT

Was ich Ihnen mit dieser Vorlesung sagen wollte:

- **Beherrschung von Technologie ist nicht ausreichend, um Probleme zu lösen**
  - **(aber sehr hilfreich dabei)**

- **Beherrschung von Technologie ist nicht ausreichend für beruflichen Erfolg**
  - **(aber einigermaßen notwendig)**

Was hat das alles mit mir zu tun? (2)



- Indische SW-Entwickler mit 3 Jahren Berufserfahrung verdienen ca. US$ 12.000/Jahr (Stand 2005)
  - http://quote.bloomberg.com/apps/news?pid=10000080&sid=ahtWCPS3laYE&refer=asia
  - Sind genauso gut ausgebildet wie Sie es sind
    - Vielleicht besser: Die indische Firma Tata TCS ist die erste der Welt, die firmenweit CMMI-Prozessreifestufe 5 erreicht hat
  - Deutsche verdienen ca. das drei- bis vierfache
- Wenn Sie nur Technologie können, ist das direkte Konkurrenz
  - Ebenso Leute in Polen, Tschechien, Ungarn, China, Russland, Ukraine, Rumänien, etc.
- Was könnten Sie demgegenüber für Vorteile bieten?
  - Sie verstehen die deutsche Kultur. (*Tun Sie das?*)
  - Sie verstehen deutsche Geschäftsprozesse (*Tun Sie das?*)
  - Sie stärken einen Weltmarktführer vor Ort (*Verstehen Sie die?*)
    - Hermann Simon: "Die heimlichen Gewinner"
    - Hermann Simon: "Hidden Champions des 21. Jahrhunderts"

Was hat das alles mit mir zu tun?

Freie Universität Berlin

- Sie werden **nicht** dafür bezahlt
  - dass Sie einen Hochschulabschluss haben
  - dass Sie etwas wissen
  - dass Sie etwas tun
  - dass Sie irgendwelche Probleme lösen
- sondern dafür
  - dass Sie solche Probleme lösen, bei denen der Nutzen der Lösung höher ist als die Kosten der Lösung

- Also lernen Sie bitte:
  1. Probleme und ihre Wichtigkeit verstehen
     - Auf allen Ebenen: Anforderungen, Entwurf, Technologieauswahl, Test, Testautomatisierung, Prozessformalisierung usw. usf.
  2. Probleme lösen
  3. Kosten und Nutzen von Technologie und Methoden abschätzen

Gütemaßstab der Softwaretechnik

- Die Softwaretechnik war für eine Software dann gut, wenn
  - der erzielte **Nutzen** viel höher ist als
  - die aufgewendeten **Kosten**
  - und das auch künftig weiter gilt (bis die SW in Ruhestand geht)
- Nutzen
  - bemisst sich nach erzielten betrieblichen **Verbesserungen** oder erzielten **Verkaufspreisen**, unter Beachtung der Sicherheit
  - hängt ab von **Funktionalität** und **Qualität**
    - und anderen Dingen (z.B. Marketing) außerhalb ihres Einflussbereichs
- Kosten
  - Kosten für Definition, Entwurf, Bau und Test
  - Kosten für (ggf.) Vermarktung, Einführung, Ausbildung, etc.
  - Laufende Kosten für Unterstützung, Pflege und Fortentwicklung

Wie optimiert man Kosten/Nutzen?

Wie maximiert man den Nutzen?

- **Wertvolle Anforderungen** aufdecken und umsetzen
- SW **hoher Qualität** produzieren

Wie minimiert man die Kosten?

- Software mit möglichst wenig Arbeitsaufwand fertig stellen
  - Schlüsseltechniken: kompetente **Anforderungsbestimmung** (spart Korrekturen), guter **Entwurf** (spart Korrekturen), **Wiederverwendung, Risikomanagement**
- Künftige Kosten vermeiden
  - keine hohen Kosten für **Defektkorrekturen**
    - erreicht durch hohe Zuverlässigkeit, also hohe Qualität
  - keine hohen Kosten bei **späteren Änderungen**
    - erreicht durch gute Wartbarkeit, ebenfalls ein Aspekt von Qualität
- Qualität spielt also eine zentrale Rolle in der SWT

Technische vs. menschliche Aspekte

- Die menschlichen Aspekte werden von vielen Softwareingenieuren und Projektmanagern gern ignoriert
  - Technische Aspekte sind aber bei Weitem nicht alles!
- 1. Softwaretechnik wird von Menschen betrieben
  - Menschliche Eigenschaften sind wichtige Treiber für die Softwaretechnik
    - Alle Beteiligte: Entwickler, Benutzer, Kunden, Manager(-innen)
  - z.B. haben Menschen Beschränkungen (etwa bei Wissen, Kurzzeitgedächtnis, Kommunikationsvermögen), die entscheidend für die Gestaltung der Methoden der SWT sind
  - Außerdem zeigen Menschen variables Verhalten: sie sind manchmal ängstlich, aggressiv, beeinflussbar, einfallslos, eitel, faul, fleißig, ignorant, kreativ, müde, mutig, souverän, redselig, schweigsam, wachsam, u.a.m.
- 2. Software wird von Menschen benutzt
  - → sozio-technisches System

Welche Arten von SWT-Situationen gibt es?

Freie Universität Berlin

Einteilung nach mehreren Kriterien:

- Nähe und Anzahl der Kunden
- Art der Benutzer, Art der Benutzung
- Größe/Komplexität der SW und des Projekts
- Wie kritisch ist (nichttechnisches) Domänenwissen?
- Müssen Näherungslösungen verwendet werden?
- Wie kritisch ist Effizienz?
- Wie kritisch ist Verlässlichkeit?

Warum will man das wissen?

- Die meisten detaillierten Methoden der Softwaretechnik sind nur für manche dieser Situationen/Softwarearten relevant
  - Und für die anderen oft uninteressant oder unbrauchbar
- Bitte immer im Auge behalten!

Schnelldurchgang durch den Stoffplan

Wichtige Themen, die wir nicht behandelt haben

1. Methoden zur Entwicklung sicherheitskritischer Systeme
2. Methoden zur Maximierung der Benutzbarkeit
3. Software-Evolution
4. Entwicklungswerkzeuge

- Dazu jetzt jeweils ganz kurz 1–2 Folien

1. Methoden zur Entwicklung sicherheitskritischer Systeme

- Bei sicherheitskritischen Systemen haben zwei Aspekte höchstes Gewicht:
  - **Verlässlichkeit** (insbes. Zuverlässigkeit und Verfügbarkeit)
  - Einhaltung von **"darf nicht"-Anforderungen**
    - Eine Sorte nichtfunktionaler Anforderungen
- Mittel zu deren Behandlung sind z.B.
  - Diverse Formen **formale**r **Spezifikation**
    - Neben Zustandsautomaten sind dies z.B.
    - Petrinetze (insbesondere bei Nichtsequentialität)
    - Mathematische Spezifikationsformen (z.B. Z)
  - **Modellprüfung**
    - Werkzeuge prüfen, ob alle Zustände einer Spezifikation sämtliche "darf nicht"-Bedingungen einhalten
  - Automatische **Codeerzeugung** oder **formale Verifikation**
    - Mit dem Code erzeuge ich zugleich den Beweis, dass er der Spezifikation entspricht

2. Methoden zur Maximierung der Benutzbarkeit

- Bei interaktiven Systemen ist der wichtigste und schwierigste Aspekt häufig die Benutzbarkeit
  - Erlernbarkeit: Wie einleuchtend ist es?
  - Bedienbarkeit: Wie nicht-umständlich ist es zu bedienen?
  - Fehlervermeidung: Wie leicht verwirrt/vertut man sich dabei?

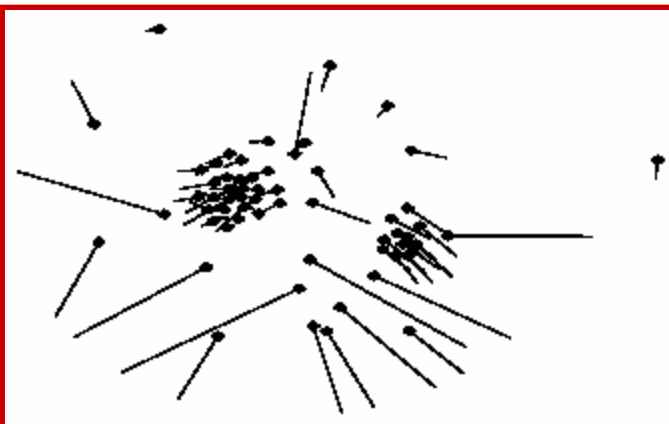
- Wichtigste Mittel zur Maximierung sind:
  - Enge Zusammenarbeit mit Benutzern
  - Anwendung bekannter Benutzbarkeits-Heuristiken
    - siehe z.B. die Benutzbarkeitsmuster
  - Ausführliches und wiederholtes Benutzbarkeitstesten

- Literatur:
  - Jakob Nielsen: "Usability Engineering", Morgan Kaufman 1994

3. Software-Evolution

- Der überwiegende Teil des Entwicklungsaufwands für ein SW-System entsteht meistens gar nicht im ursprünglichen Projekt
  - sondern durch Pflege (Erweiterungen, Anpassungen) und Wartung (Defektkorrekturen) später
  - Diese Phase des SW-Lebenszyklus nennt man SW-Evolution

- 2 Hauptprobleme dabei:
  1. Die beauftragten Leute kennen die SW nicht genau von innen
     - Es ist weniger Wissen über die konkreten Anforderungen und den zugehörigen Entwurf verfügbar als während des Projekts
  2. Der Entwurf der SW verfällt bei aufeinander folgenden Änderungen allmählich immer weiter
     - weil oft eine "saubere" Lösung angesichts des geringen Umfangs der Änderung aufwändiger ist
     - weil oft Wissen über den Entwurf fehlt und deshalb die "richtige" Lösungsweise übersehen wird



Lutz Prechelt, IEEE Trans. on Software Engineering 27(1):1-12

Software-Evolution (2)

Konsequenzen:

1. Es ist sehr viel Aufwand für das Verstehen existierender SW nötig
   - Deshalb ist gute Dokumentation so wichtig
2. Die Software wird irgendwann allmählich instabiler
   - Jede Änderung erzeugt potentiell neue Mängel
   - z.B. wird bei Korrektur eines Defekts ein neuer Defekt eingefügt
   - erst recht enthalten neue Funktionen besonders viele Defekte

Lösungsansätze:

1. Werkzeuge für das SW-Verstehen (reverse engineering)
   - Für Sprachen wie Java z.T. in brauchbarer Qualität verfügbar
   - Für Programme in z.B. C teilweise sehr schwierig
2. Werkzeuge für das Reengineering
   - d.h. das Umstrukturieren, um die Struktur wieder zu verbessern

4. Entwicklungswerkzeuge

- Früher erfolgte SW-Entwicklung weitgehend von Hand
  - Code schreiben mit Texteditor
  - Diagramme malen ggf. mit Zeichenprogramm
  - Wichtigstes "Werkzeug" war der Übersetzer
- Das wird heute zunehmend seltener. Wir benutzen:
  - Ausgefeilte Code-Entwicklungsumgebungen
  - Diverse Arten von Codegeneratoren, Middleware
  - Versions- oder Konfigurationsverwaltung, Testautomatisierung
  - Modellierungswerkzeuge
  - Prüfwerkzeuge, Analysewerkzeuge etc.
- All dies ist (bei klugem Einsatz) gut und nützlich
- Leider ist Wissen über Werkzeuge recht kurzlebig
  - Orientieren Sie sich regelmäßig, was es so gibt
  - Wenn Sie etwas verwenden, lernen Sie es gut zu beherrschen
    - Die meisten Leute nutzen viel zu wenig von dem, was drin steckt
    - z.B. in Eclipse: die Navigations- und die Refactoring-Funktionen

Was erwartet Sie in der Praxis?

- Leider ist gute, klug eingesetzte Softwaretechnik eher selten
- Zwei Tendenzen herrschen vor:
  1. Wenig Methodeneinsatz, hoher Chaosfaktor, Herumwursteln
     - Vorherrschend bei kleineren Organisationen
  2. Hoher Methodeneinsatz, aber überzogene Formalisierung und Bürokratisierung, fragwürdiges Kosten/Nutzen-Verhältnis
     - Eher bei großen Organisationen

- Woran liegt das jeweils? Bei
  1. Mangelnde Disziplin; sehr kurzfristige Denkweise; kein Nachweis über die Wirksamkeit der Methoden verfügbar ("not invented here")
     - Mangel an quantitativer Analyse des Prozesses
  2. "Cover your ass"-Einstellung; Sieg der SJ-Temperamente

Wie sollten Sie damit umgehen?

- Egal, welche der beiden Fraktionen Sie erwischen: Machen Sie sich klar, Verbesserungen gibt es nur, wenn sich jemand darum bemüht
  - Warum also nicht Sie?
  - Falsche Antwort: "Weil ich nur ein kleines Rädchen bin." (Wodurch sind wohl die "Großen" groß geworden?)

- Wenn Sie eine Verbesserung durchsetzen wollen, gibt es zwei Wege:
  - Es einfach tun, darüber sprechen und hoffen, dass es sich durchsetzt
    - Funktioniert bei sehr offensichtlichen Verbesserungen oft gut
  - Zahlen zur Wirtschaftlichkeit vorlegen
    - oft sehr schwierig, aber manchmal auch gut machbar, z.B. Durchsichten-Nutzen per Experiment nachweisen
      - Jalote, Haragopal: "Overcoming the NAH syndrome for inspection deployment", 20th Intl. Conf. on SW Eng., IEEE, 1998

Eigenverantwortung!

Was sollten Sie können?

- Methodik
  - Genug, um in mäßig komplexen Projekten sicher agieren zu können (Krisen vermeiden, befriedigende Qualität garantieren)
- Technologie
  - Solides, breites Wissen über die Grundkonzepte
  - Ausreichendes Detailwissen über konkret benutzte Teilbereiche
    - Nicht für jede Technologie, sondern nur für regelmäßig benutzte
    - Im Bedarfsfall nicht pfuschen, sondern sauber dazulernen!
- Domäne
  - Mindestens eine Anwendungsdomäne (ggf. eine technische) gut kennen:
    - Anforderungen oft selbst entdecken können
    - Anforderungen stets verstehen und einordnen können
- **Vor allem sollten Sie Ihr Leben lang kontinuierlich hinzu lernen**

Lebenslanges Lernen: Wie? Wo?

Freie Universität Berlin

- Allgemein, breit:
  - Gesellschaft für Informatik www.gi-ev.de
    - Berufsstandsvereinigung: Regionalgruppen, Interessengruppen, Tagungen, *Informatik-Spektrum*, *Computer-Zeitung*
  - Oder international: ACM oder IEEE Computer Society
    - www.acm.org, www.computer.org
- Technologiespezifisch:
  - Benutzergruppen zu Herstellern, Produkten, Produktarten
  - Lose Communities im Web
- Methodik:
  - Interessengruppen z.B. zu Projektmgmt, Entwicklungsprozessen, Architektur, Testen, etc.
  - meist innerhalb der Informatik-Gesellschaften (s. oben), z.T. auch lose im Web

Danke!

Viel Erfolg!