Vorlesung "Softwaretechnik"

Schluss

Lutz Prechelt

Freie Universität Berlin, Institut für Informatik

- Wiederholung einiger Aussagen der 1. Vorlesung
  - Worauf kommt es an?
- Was erwartet mich in d. Praxis?
- 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? (Architektur, Modulzerlegung)
- **Bauen** (Implementierung)
- **Validieren** der Lösung (Analytische Qualitätssicherung)
  - Hat die SW die erwünschten Eigenschaften?
- **Verwalten** des Vorgehens (Management)
  - Den ganzen Prozess, mit dem obige Aufgaben gelöst werden, gestalten (z.B. planen, Qualitätsmängeln vorbeugen etc.)

Alle 6 Aufgaben sind eng miteinander verwoben

DIE NACHRICHT

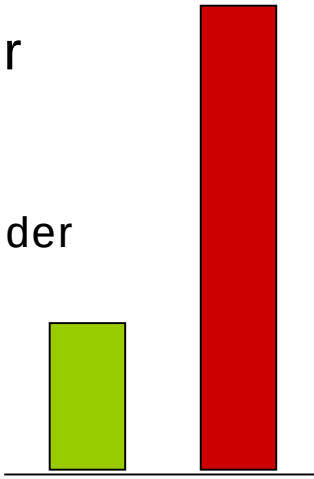
Was ich Ihnen mit dieser Vorlesung sagen wollte:

- **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 verdienen ca. US$ 16.000/Jahr
  https://www.averagesalarysurvey.com/software-developer/india
  - Sind genauso gut ausgebildet wie Sie es sind
    - Vielleicht besser: Die indische Firma Tata TCS war die erste der Welt, die firmenweit CMMI-Prozessreifestufe 5 erreicht hat
  - Deutsche verdienen ca. das 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?*)
    - "Hidden Champions" (Hermann Simon)

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**

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?

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!

Was erwartet Sie in der Praxis?

- Leider ist gute, klug eingesetzte Softwaretechnik immer noch nicht der Normalfall
- Es herrschen statt dessen nicht selten vor:
  - **A.** Wenig Methodeneinsatz, hoher Chaosfaktor, Herumwursteln
    - angeblich agil heißt noch lange nicht agil
    - Vorherrschend bei kleineren Organisationen ODER
  - **B.** Hoher Methodeneinsatz, aber überzogene Formalisierung und Bürokratisierung, fragwürdiges Kosten/Nutzen-Verhältnis
    - Eher bei großen oder sicherheitskritischen Projekten

- Woran liegt das jeweils? Bei
  - **A.** Mangelnde Disziplin; sehr kurzfristige Denkweise
  - **B.** "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

Was erwartet Sie in der Praxis? (2)

Bei einer recht guten Organisation (Microsoft):

- Wichtigste Faktoren f. Zufriedenheit aus Sicht von Entwickler_innen:
  - Manager_in
  - subjektive Produktivität
  - "rewards"
  - Teamkultur
  - meine Fähigkeiten kommen gut zur Geltung

(unwichtigster von 40: Fortbildungen zu Werkzeugen)

Fast alles Prozessfaktoren!

Quelle: Towards a Theory of SW Developer Job Satisfaction and Perceived Productivity

- Wichtigste Problembereiche aus Sicht von Entwickler_innen:
  - schlechte Architektur
  - Altcode
  - (weitere Facetten von Komplexität)

  } OK

  - viele Unterbrechungen
  - Manager

  } Doof!

- Folgerung: Suchen Sie sich eine gute Kultur!

Schnelldurchgang durch den Stoffplan

Wichtige Themen, die wir nicht behandelt haben

Freie Universität Berlin

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

- Dazu jetzt ganz kurz je eine Folie:

1. Methoden zur Entwicklung sicherheitskritischer Systeme

Freie Universität Berlin

- Bei sicherheitskritischen Systemen (ob Safety oder Security) haben zwei Aspekte höchstes Gewicht:
  - f. Safety: **Verlässlichkeit** (v.a. Zuverlässigkeit + Verfügbarkeit)
  - f. beides: Einhaltung von **"darf nicht"-Anforderungen**

- Mittel für Verlässlichkeit
  - Diverse Formen **formale**r **Spezifikation**
    - z.B. Zustandsautomaten, Petrinetze, VDM, Z
  - **Modellprüfung**
  - Automatische **Codeerzeugung** oder **formale Verifikation**
- Mittel für Security (Schutz)
  - geeignete **Entwicklungsprozesse** mit vielen Elementen
    - siehe z.B. OWASP, Microsoft Security Engineering portal und Lehrveranstaltungen von Prof. Jörn Eichler
    - insbes. viele Durchsichten mit Fokus auf typische Sorten von Sicherheitslücken; Verständnis dieser typischen Sorten

2. Methoden zur Maximierung der Benutzbarkeit

Freie Universität Berlin

- Bei interaktiven Systemen ist der wichtigste und schwierigste Aspekt häufig "User Experience":
  - Erlernbarkeit: Wie einleuchtend ist es?
  - Angemessenheit: Wie einfach ist es, seine Aufgaben zu lösen?
  - Fehlervermeidung: Wie leicht verwirrt/vertut man sich dabei?

- Wichtigste Mittel zur Maximierung sind:
  - Zusammenarbeit mit Benutzer/inne/n, **User-Centered Design**
    - plus iterative Entwicklung
  - Anwendung von Benutzbarkeits-Heuristiken
    - siehe z.B. die Benutzbarkeitsmuster

- an der FU Berlin:
  - Lehrveranstaltungen von Prof. Claudia Müller-Birn

3. Software-Erosion

- Das Hauptproblem vieler Teams ist eine schwer verständliche und schwer änderbare Codebasis
  - Die Architektur ist im Laufe der Jahre erodiert
  - Das passiert, wenn man bei Änderungen zu oft
    - Abkürzungen nimmt (Projektmanagement!) oder
    - die Architektur nicht richtig verstanden hatte (Dokumentation!)

- Mittel dagegen:
  - Angemessenes Entwicklungstempo: Kein Zeitdruck!
  - Kontinuierliches "Aufräumen": Refactoring
    - z.B. Pfadfinderregel: Hinterlasse die Codebasis stets in einem besseren Zustand als Du sie vorgefunden hast.

Eick et al.: Does code decay?

4. Entwicklungswerkzeuge

- Viele Werkzeuge sind heute gängig und verbreitet, z.B.
  - Compiler, Build-Werkzeuge, Versionsverwaltung
  - IDEs (z.B. Eclipse, IntelliJ, Visual Studio Code)
  - Testframeworks, CI/CD-Werkzeuge
- Diese werden künftig immer weiter hinzulernen:
  - IDEs bekommen bessere Funktionen für Programmverstehen
  - und für Refactoring.
  - Compiler oder separate Prüfwerkzeuge lernen mehr und mehr semantische Korrektheitsprüfungen.
- 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 IDEs: die Navigations- und die Refactoring-Funktionen

Eigenverantwortung!

Was sollten Sie können?

Freie Universität Berlin

1. Methodik
   - Genug, um in mäßig komplexen Projekten sicher agieren zu können (Krisen vermeiden, befriedigende Qualität garantieren)
     - **Augen auf für die sozialen Faktoren der SW-Entwicklung!**
2. 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!
3. 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?

- Gesellschaft für Informatik (GI) www.gi-ev.de
  - mit z.B. Regionalgruppen, Interessengruppen, Tagungen
  - Interessengruppen z.B. zu Projektmgmt, Entwicklungsprozessen, Architektur, Testen, etc.
- Oder international: ACM oder IEEE Computer Society
  - acm.org , computer.org
- Konferenzen
- Lose Communities, z.B. via meetup.com
  - in Berlin sehr ergiebig!

Danke!

Viel Erfolg!

Und nochmal der Hinweis:
Suchen Sie sich einen Arbeitsplatz mit einer guten Kultur!