Software-Entwicklung

Geschichte der Programmierung

Aufgaben von, Anforderungen an Programme mit der Zeit verändert

1. Programmierung über Lochkarten
   z.B. für Rechenaufgaben
2. maschinennahe Programmierung
   auch wichtig wegen Geschwindigkeitsverlusts durch Compilation
3. höhere Programmiersprachen
   geeignet für größere Programme
4. ???
   komplexe Software-Systeme

Die Software-Krise

in den 1960er Jahren begannen Kosten für Software die Kosten für Hardware zu übersteigen

davor: Computer nicht sehr leistungsfähig, keine großen Programme nötig

ab dann: Computer leistungsfähig, Software muss mithalten

Begriff *Software-Engineering* / *Softwaretechnik* wurde geprägt

generelle Ziele der Software-Entwicklung:

- Korrektheit
- Effizienz
- Wartbarkeit
- Wiederverwertbarkeit

Softwaretechnik

beschreibt Prozess des Herstellens von Software

mehr als nur Erstellen eines Programms, sondern u.a.

- Planung
- Analyse
- Entwurf / Modellierung
- Programmierung
- Validierung
- Dokumentation

Planung

Planung eines Softwareprojekts beinhaltet normalerweise u.a.

- Aufwandsschätzung
  - Zeit
  - Personal
  - Budget
  - ...
- Planung der Vorgehensweise

hier keine Modelle der Aufwandsschätzung, aber zeitliche Planung wird von Gruppen erwartet

Zeitliche Planung

beachte zeitlichen Rahmen

- Endabnahme in der Woche vom 9.02.09–13.02.09
  voll funktionsfähiges System, sinnvoll dokumentierter Quellcode, etc.
- Wettbewerb am 5.02.09
- Testabnahme in der Woche 15.12.08–19.12.08
  funktionsfähiges aber nicht unbedingt vollständiges System, sinnvoll dokumentierter Quellcode, etc.

definiert zwei Phasen der Software-Entwicklung

jede Gruppe legt Tutor sobald wie möglich Planung für verbleibenden Zeitraum vor (was wird wann gemacht?)

Vorgehensweisen

werden in Vorgehensmodellen festgelegt

Vorgehensmodelle machen in verschiedener Detailliertheit
Vorgaben zum Ablauf der einzelnen Prozesse (wie, wann)

verschiedene Typen von Vorgehensmodellen, z.B.

- eher Philosophie
  - V-Modell
  - Modellgetriebene Software-Entwicklung
  - Agile Software-Entwicklung
  - Design by Contract
- Prozessmodelle, z.B.
  - Wasserfallmodell
  - Spiralmodell

Vorgehensweisen im Praktikum

nicht jedes Vorgehensmodell in jeder Situation anwendbar

z.B. hier: Kunde nicht wirklich vorhanden (Aufgabensteller, Tutoren?)

Teilnehmer sollen über Existenz von Vorgehensmodellen informiert sein

Vorgehensmodelle im Detail mit Diskussion $\rightsquigarrow$ Vorlesung Softwaretechnik

Gruppen sollen hier vermitteltes Wissen über Vorgehensmodelle einsetzen (z.B. Zerlegung des Entwicklungsprozesses in Teilprozesse)

Agile Software-Entwicklung

kein einzelnes Prozessmodell, sondern Entwicklungsphilosophie

hier nicht unbedingt Empfehlung, schon gar nicht Vorschrift

Philosophie definiert drei aufeinander aufbauende Ebenen

- *agile Werte*
  was bei der Software-Entwicklung wichtig sein sollte
- *agile Prinzipien*
  generelle Vorschriften für alle Teilprozesse um die Werte zu respektieren
- *agile Methoden*
  Beschreibung der Ausführung von Teilprozessen nach den Prinzipien

Agile Werte

- Wichtiger als vordefinierte Prozesse und vorgefertigte Tools sind die beteiligten Individuen und deren Miteinander.
- Wichtiger als ausführliche Dokumentation ist eine funktionierende Software.
- Wichtiger als nach Vertrag zu arbeiten ist die Zusammenarbeit mit dem Kunden.
- Wichtiger als einem Plan zu folgen ist es, flexibel auf Änderungen reagieren können.

allgemein: Verzicht auf starre Regeln

Agile Prinzipien

1. Kunde durch frühes und häufiges Liefern zufrieden stellen
2. sich ändernde Anforderungen sind willkommen
3. funktionierende Software häufig abliefern
4. tägliche Zusammenarbeit zwischen Entwicklern und Managern
5. Projekte von motivierten und zufriedenen Mitarbeitern durchführen lassen
6. Information durch Konversation übermitteln
7. Erfolg wird an funktionierender Software bemessen
8. nachhaltige Entwicklung
9. ständige Aufmerksamkeit auf technische Exzellenz und gutes Design
10. Einfachheit ist essentiell
11. selbst-organisierende Teams erbringen beste Designs, etc.
12. Selbstreflektion im Team in regelmäßigen Abständen mit anschließender Verbesserung

Agile Methoden

Methoden sollen Prinzipien in Entwicklungsprozesse einbringen

Aversion gegen starre Pläne, besser Fokussierung auf das, was gebraucht wird, statt dem, was geplant ist

Beispiele für agile Methoden:

- Programmieren zu zweit
- testgetriebene Entwicklung

Abschluss: agile Software-Entwicklung erfreut sich wachsender Beliebtheit und ist in vielen Fällen erfolgversprechend

Validierung

Erinnerung: Korrektheit von Software ist eines der Ziele ihrer Entwicklung

imminente Frage: wann ist ein Programm korrekt?

offensichtliche Antwort: wenn es den Anforderungen genügt

Antwort nicht sehr befriedigend ("wann genügt ein Programm seinen Anforderungen?")

Ansatz: formalisiere Anforderungen als *Eigenschaften* von Programmen

Der Satz von Rice

Typ einer Eigenschaft ist lediglich *Menge von Programmen*

(Eigenschaft E kodiert als Menge aller Programme, die E haben)

Eigenschaft trivial: $E = \emptyset$ oder $E =$ Menge aller Programme

Satz: Sei E eine nicht-triviale Eigenschaft von Programmen. Dann gibt es keinen Algorithmus, der zu einem vorgegebenen Programm P entscheidet, ob $P \in E$ gilt oder nicht.

Konsequenz: Validierung von Programmen nicht automatisierbar

Validierung

Validierung dennoch wichtig, denn z.B.

- Software soll fehlerfrei sein
- Kunde kauft nicht gerne Katze im Sack

verschiedene Methoden zur Validierung von Software

- Validierung per Hand, z.B. Theorembeweisen
- proof carrying code
- automatische Validierung eingeschränkter Programme, z.B. durch Abstraktion
- Testen
- ...

Testen

Testen = exemplarische (also nicht vollständige) Validierung, nicht nur bzgl. Fehlerfreiheit, sondern auch bzgl. genereller Spezifikation

Vorteil von Testen: nicht eingeschränkt durch Programmgröße

Nachteil: kann nur Fehler aufdecken, aber keine Fehlerfreiheit garantieren

in vielen Vorgehensmodellen: Validierung (z.B. durch Testen) nach Programmierung oder zeitlich unabhängig davon

in testgetriebener Entwicklung: erst Tests definieren, dann Programmcode erzeugen

vorliegende Tests können auch als Dokumentation gesehen werden

Testarten

man unterscheidet drei Arten von Tests bzgl. Kenntnis des Codes:

- white-box test: Code ist bekannt (z.B. wegen Testen nach Programmieren)
- black-box test: Code ist unbekannt, stattdessen nur Spezifikation
- grey-box test: Code ist teilweise bekannt, z.B. weil *noch nicht* geschrieben

man unterscheidet zwei Arten von Tests bzgl. Granularität:

- Systemtests
- unit tests: einzelne Komponenten werden getestet
  Annahme: Gesamtsystem korrekt, wenn alle Komponenten korrekt sind

unit tests haben größere Chance, Fehler aufzudecken, erfordern aber detaillierte Spezifikation

Testgetriebene Entwicklung

benutzt grey-box unit tests

Codegenerierung erfolgt in Iteration von Phasen

1. identifiziere nächste zu integrierende Funktionalität
2. schreibe Tests dafür
3. solange diese Tests fehlschlagen: schreibe/repariere Code dafür
4. vereinfache Gesamtcode durch Abstraktion, Aufräumen, etc.
5. wiederhole Tests