Vorlesung „Softwaretechnik"

# Software-Entwurf

Dr. Lutz Prechelt

- Ziele
- Entwurf vs. Architektur
- Vorgehen



## Definition: Software-Entwurf (SW Design)

- Festlegung der Struktur eines Softwaresystems
  - Welche Teile gibt es?
  - Was genau sind deren Aufgaben (Schnittstellen-Definition)?
  - Welche Beziehungen bestehen zwischen ihnen?
- Meist ist keine genaue Unterscheidung zwischen Vorgang (das Entwerfen) und Ergebnis (der fertige Entwurf) nötig



## Entwurf vs. Architektur

- Verbreiteter Sprachgebrauch:
  - Architektur = grobe Strukturfestlegungen
  - Entwurf = detaillierte Strukturfestlegungen
  - Implementierung = endgültige Struktur
- Besserer (weil präziserer) Sprachgebrauch:
  - Architekturaussage
    - bezieht sich auf alle Teile (Elemente) des Entwurfs/Systems
    - kann auf vielerlei Art realisiert werden
  - Entwurfsaussage
    - bezieht sich nur auf bestimmte Teile des Entwurfs/Systems
    - kann auf vielerlei Art realisiert werden
  - Implementierungsaussage
    - bezieht sich nur auf bestimmte Teile des Entwurfs/Systems
    - kann nur auf eine Art realisiert werden



## Entwurf: Wann?

Bezüglich Definitionsphase:

- Traditionelles Verständnis (Wasserfall-Modell):
  - Vor Beginn des Entwurfs müssen alle Anforderungen definiert und verstanden sein (Änderungen allenfalls noch im Verlauf des Entwurfs erlaubt).
- Moderneres Verständnis (Evolutionäres Modell):
  - Vor Beginn des Entwurfs sollten zumindest einige der Anforderungen einigermaßen verstanden sein
- Evolutionäres Modell ist erheblich realistischer. ABER:
- Sehr große Projekte sind im Evolutionären Modell sehr schwierig zu beherrschen

Bezüglich Implementierungsphase:

- Vor Beginn der Implementierung sollten die betreffenden Elemente des Entwurfs definiert sein.

## Entwurf: Ziele

1. Beherrschbarkeit
   - Reine Komplexität
   - Analysierbarkeit, Testbarkeit
   - Wartbarkeit
   - Parallele Implementierung
2. Wiederverwendung und Wiederverwendbarkeit
   - Wiederverwenden statt Bauen
   - Wiederverwenden statt Hinzubauen
   - Wiederverwenden statt Ähnliches zu bauen
   - Produktfamilien
3. Effizienz [oftmals kaum problematisch]
   - Laufzeit (Durchsatz, Antwortzeiten)
   - Speicherbedarf (Hauptspeicher, persistenter Speicher)
   - Netzkapazität



## Beherrschbarkeit: Reine Komplexität

- Große Softwaresysteme haben so viele Einzelteile, dass niemand sie mehr komplett überschauen kann
  - Diese Komplexität muss im Entwurf beherrscht werden
- Lösungsansätze: Abstraktion, Hierarchisierung
- Entwurf muss klären:
  - Welche Teile sind für was zuständig?
  - Und für was nicht?
  - Welche Teile decken gemeinsam welche Anforderungen ab?
  - Und wie funktioniert das Zusammenspiel, damit das klappt?



## Beherrschbarkeit: Analysierbarkeit

- Es ist zu riskant, ein großes System erst komplett zu bauen, um dann zu prüfen, ob es alle Anforderungen abdeckt.
- Deshalb muss der Entwurf die Analyse erlauben, ob (und wie und wo) jede Anforderung von ihm erfüllt wird.
- Ansatz: Rückverfolgbarkeit von Anforderungen aus dem Entwurf



## Beherrschbarkeit: Testbarkeit

- Durch die vielen Einzelteile lassen sich moderne Softwaresysteme nicht sinnvoll im Ganzen testen
  - Zu schwierig, alle Defekte zum Vorschein zu bringen
  - Wenn entdeckt, sind Defekte zu schwierig zu lokalisieren
  - Ihre Behebung erzeugt oft neue Defekte
- Klare Zuweisung von Aufgaben an die einzelnen Systemteile ermöglicht Testen einzelner Teile oder von kleineren Teilegruppen
  - Erfüllt dieses Teil (diese Teilegruppe) die vorgesehenen Aufgaben
  - Was selbst dann noch schiefgehen kann: Zusammenspiel der Teilegruppen

## Beherrschbarkeit: Wartbarkeit

- Auch nach längerer Zeit müssen noch Veränderungen vorgenommen werden
  - Oft durch Personen, die nicht am Bau beteiligt waren
- Teilaufgaben dabei:
  1. Betroffene Region des Systems finden
  2. Deren Konstruktionsprinzip verstehen
  3. Änderung entwerfen
  4. Änderung einbauen
  5. Änderung und ganze betroffene Region neu testen
- Schritt 2 ist nur mit Hilfe von Entwurfsdokumentation möglich
  - Was tut diese Region? Welche anderen Teile benutzt sie?
  - Wie ist diese Region aufgebaut? Warum?
- Alternative: Änderung als „Erker" anbauen
  - Zerrüttet Entwurfsstruktur alsbald



## Beherrschbarkeit: Parallele Implementierung

- Wenn mehr als eine Person das System implementieren muss, wird sehr bedeutsam, wie viel Kommunikation zwischen den Beteiligten dabei nötig ist
  - Schlimmstenfalls muss jeder mit jedem sprechen: Kommunikationsbedarf steigt quadratisch mit der Teamgröße!
- Ein guter Entwurf minimiert den Kommunikationsbedarf, weil relevante Fragen von ihm schon beantwortet werden.
  - d.h. offen gelassene Fragen betreffen möglichst jeweils nur 1 Person
- Lösungsansatz: Abstraktion, Hierarchisierung



## Lösungsansätze für Beherrschbarkeit

Prinzip der Modularisierung:

- Zerlege System in Teile, die sich einzeln verstehen und bauen lassen
  - Zerlege ggf. Teile wiederum in Teile
  - Prinzip der Hierarchisierung
- Jedes Teil braucht dazu wohldefinierte Rechte und Pflichten: Schnittstellen (interfaces), auch genannt Verträge („design by contract")
  - Eine Schnittstelle besteht aus
    - einer Syntax (z.B. interface-Deklaration in Java oder einzelne Methodensignatur)
    - einer Menge von verlangten Vorbedingungen
      - Was erwartet das Teil von seiner Umgebung?
    - einer Menge von zugesicherten Nachbedingungen
      - Was leistet das Teil für seine Umgebung?
  - Prinzip der Abstraktion



## Wiederverwendung vs. Wiederverwendbarkeit

- Wiederverwendung:
  - verwende existierendes Element statt neues zu erfinden
  - sofortiger Nutzen
- Wiederverwendbarkeit
  - Erfinde Element so, dass es später wiederverwendet werden kann
  - Verzögerter Nutzen:
    - Geringerer Aufwand für spätere Produktversionen/Wartung
    - Evtl. Nutzung durch andere Projekte der selben Firma
    - Kann evtl. separat verkauft werden
    - Kann evtl. Verkaufsargument für dieses System sein
      - z.B. falls Erweiterungsschnittstellen für Kunden offen gelegt.

## Wiederverwenden statt Bauen

- Verwendung geeigneter fertiger Elemente
  - spart Aufwand
  - verbessert Qualität
- Problemfelder:
  - wie finde ich geeignete Elemente?
  - sind sie wirklich geeignet?
    - funktionale Abdeckung, Flexibilität, Qualität
  - wie setze ich sie richtig ein?
- Größtes Verbesserungspotential innerhalb der Softwaretechnik überhaupt liegt in diesem Bereich!



## Wiederverwenden statt Bauen (2)

Im Großen (Gesamtsystem- oder Subsystemebene):

- Heute oft wichtigste Entwurfsentscheidung: *„Make or buy" (Selber bauen oder Kaufen)*
  - z.B. CRM-System f. kleinere Firmen
  - z.B. Excel als Plattform für Statistiksoftware

Im Kleineren (Elementaraufgaben):

- Vielzahl von Bibliotheken u.ä. für verschiedene Aufgabenbereiche verfügbar
  - z.B. GUI-Bibliothek, XML-Parser, Krypto-Funktionen, ...
- In beiden Bereichen große Fortschritte in den letzten Jahren
- Bei der mittleren Granularität hapert es noch



## Wiederverwenden statt Hinzubauen

- Ein Entwurf sollte möglichst so gestaltet sein, dass künftige Erweiterungen an der Software keine großen Umbauten verlangen
  - Flexibilität der Entwurfsstrukturen
  - Evtl. auch Flexibilität der implementierten Einzelteile
- Eine erfolgreiche Software verursacht den überwiegenden Teil ihrer Kosten erst nach Fertigstellung von Version 1.



## Wiederverwenden statt Ähnliches zu Bauen

- Oft werden an verschiedenen Stellen in einer Software ähnliche (oder sogar gleiche) Funktionen benötigt
- Diese Stellen sollten im Entwurf identifiziert werden und mit einem gemeinsamen Entwurfselement gelöst
- Beispiel: Mehrere MS-Office-Versionen lang gab es in Word, Powerpoint und Excel separate (und unterschiedlich funktionierende) Fassungen des „Datei öffnen"-Dialogs.
- Ist meist nur ein Problem, wenn Entwurf von mehreren nicht eng zusammen arbeitenden Leuten gemacht wird.
- In der Wartung allerdings schwieriger zu lösen
  - z.B. eine Funktion ist schon vorhanden, müsste aber verallgemeinert werden.

## Spezialfall: Produktfamilien

- Heute wird oftmals gleich eine ganze Familie von Softwaresystemen auf einmal entworfen
  - z.B. die SW für eine ganze Baureihe von Handytypen
- Zunächst wird nur 1 Produkt gebaut, aber die Wiederverwendung für spätere ist wesentliches Entwurfsziel



## Lösungsansätze für Wiederverwendung

- Wiederverwendung auf verschiedenen Ebenen:
  1. Laufende Systeme (Dienste)
  2. Anforderungen (Anforderungsmuster)
  3. Architekturen (Architekturmuster)
  4. Teilentwürfe (Entwurfsmuster)
  5. Implementierungen (Binärcode, Komponenten)
  6. Implementierungsteile (Quellcode)
- Maximaler Nutzen bei *gleichzeitiger* WV mehrerer Ebenen
- Anmerkung:
  - Jede Methodenlehre (z.B. die Softwaretechnik) ist Wiederverwendung
  - Aber überwiegend auf der Prozess-Ebene, nicht der Produkt-Ebene



## Lösungsansätze für Wiederverwendung (2)

- Abstrahiere möglichst von Dömänenspezifika
  - Vielleicht gibt es bereits eine Lösung für ein allgemeineres Problem
- Trenne scharf fachliche Logik von technischen Aspekten
  - Oft gibt es zumindest bereits Lösungen für die technischen Aspekte
- Suche nach Lösungen für horizontale Teilbereiche
  - z.B. GUI, Datenhaltung



## Lösungsansätze für Wiederverwendbarkeit

- Abstrahiere möglichst von Dömänenspezifika
- Trenne scharf fachliche Logik von technischen Aspekten
  - z.B. eine bestimmte Prozesssteuerung von einem Prozessteuerungsmechanismus
- Trenne scharf Schichten
  - z.B. GUI, Prozesssteuerung, Fachdienste, Datenhaltung
- Minimiere Importschnittstellen
  - ggf. durch weitere Zerlegung der Teile
- Baue Rahmenentwürfe (frameworks):
  1. definiere sehr allgemeine Schnittstellen
  2. verwende wo möglich nur die abstrakte Schnittstelle
  3. löse konkretes Problem durch separate Implementierungen dieser Schnittstellen
  - Viele Beispiele hierfür bei Entwurfsmustern

## Effizienz

Entscheidende Effizienzfaktoren sind heute zumeist

- Übertragenes Datenvolumen (Dateien, DB, Netz)
- Anzahl kurzlebiger Objekte
- Unnötige Wiederholungen / Caching

Diese Faktoren sind in der Entwurfsphase analysierbar:

- Überschlagsrechnung als Test ob Leistung kritisch ist
- Ggf. Einzelrechnungen um Engpässe aufzuspüren
- Änderung der Entwurfsstruktur, um Engpässe zu beseitigen
  - Vergleich mehrerer Entwurfsalternativen
    - z.B. Datenübertragungen einsparen, Objekte durch Stellvertreter ersetzen, Ergebnisse aufbewahren
  - Siehe auch Implementierungsphase



# Danke!