1. Einführung

1.1 Entwerfen – Warum?

- ☆ Beim Arbeiten im Kleinen nicht oder nur ansatzweise (Detailentwurf)
- ☆ Größere Software braucht einen systematischen Aufbau
  - ✧ Verstehen: Software >> 1 Kopf
  - ✧ Arbeit verteilen: Software-Entwicklung ist Teamarbeit
  - ✧ Einbetten in vorhandene Software: Schnittstellen und Wechselwirkungen verstehen
  - ✧ Verteilen auf Hardware-Komponenten, geographische Verteilung
  - ✧ Lokalisieren: Lokal ändern können, Auswirkungen von Erweiterungen und Ergänzungen abschätzen/begrenzen

- ➪ Aufbau im Großen: Software-Architektur, Konzept
- ➪ Aufbau im Detail: Software-Entwurf (und Programme)

1.2 Definitionen

Architektur (architecture) – Die Organisationsstruktur eines Systems oder einer Komponente. (IEEE 610.12)

Entwurf (design) – 1. Der Prozess des Definierens von Architektur, Komponenten, Schnittstellen und anderen Charakteristika eines Systems oder einer Komponente.
2. Das Ergebnis des Prozesses gemäß 1. (IEEE 610.12)

Software – Die Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Rechnersystems zu tun haben. (IEEE 610.12)

1.3 Entwurfsprinzipien

Systematischer Aufbau: Die Grundlage guten Entwurfs

- **Strukturen** (Module, Prozesse, Kommunikation...):
  - Gliederung in Komponenten und Interaktion zwischen Komponenten
- **Abstraktionen** (Komposition, Benutzung, Generalisierung, Aspekte)
  - Systematische Vergröberung und Verfeinerung nach verschiedenen Kriterien
  - Verständnis großer Zusammenhänge unter Weglassung der Details
  - Verständnis eines Details unter Weglassung/starker Vergröberung des Rests
  - Systematischer Zusammenhang zwischen Übersichten und Detailsichten

Modularität: Divide et impera oder *Das* Entwurfsprinzip schlechthin

- ☆ Gliederung der Gesamtlösung in kleinere, überschaubare Teillösungen
- ☆ Nicht irgendwie gliedern, sondern:
  - Modul = in sich geschlossene Einheit
  - Verwendung erfordert keine Kenntnisse über inneren Aufbau
  - Kommunikation mit Umgebung ausschließlich über Schnittstelle
  - Rückwirkungsfreie Änderbarkeit im Modulinnern
  - Korrektheit ohne Kenntnis der Einbettung prüfbar
- ☆ zwingend bei Entwurf großer Systeme

Das Geheimnisprinzip (Information Hiding): Das Fundamentalprinzip zur Gliederung komplexer Systeme

- ☆ Modularisierungsprinzip: Kapselung von Entwurfsentscheidungen
- ☆ Modulverwender wissen, welche Entwurfsentscheidungen ein Modul implementiert, aber nicht wie.
- ☆ Typische Arten von Entwurfsentscheidungen:
  - ✧ wie eine Funktion realisiert ist
  - ✧ wie eine Datenstruktur aufgebaut ist und wie sie bearbeitet werden kann
  - ✧ wie Leistungen Dritter genutzt werden

Zusicherungen und Verträge: Die starken Geschwister des Geheimnisprinzips

- ☆ Zusicherungen definieren das Leistungsangebot eines Moduls
  - ✧ Voraussetzungen – was vorher erfüllt sein muss
  - ✧ Ergebniszusicherung – was nachher erfüllt ist
  - ✧ Invarianten – was nicht verändert werden darf
  - ✧ Verpflichtungen – mit der Verwendung übernommene Pflichten

- ☆ Verträge regeln die Zusammenarbeit (z. B. Delegation von Teilaufgaben) zwischen Modulen ("Design by Contract")

Qualität: Du sollst nicht schludern

Ein guter Entwurf ist:

- ☆ Effektiv: erfüllt die Vorgaben, löst das Problem des Auftraggebers
- ☆ Robust
- ☆ Wirtschaftlich: gebrauchstauglich, kostengünstig, mehrfachverwendbar/mehrfachverwendet

- ➪ Erfordert kontinuierliche Prüfung

Ästhetik und Eleganz: Was gut ist, ist auch schön

- ☆ Gestaltet statt geworden
- ☆ Einfach und klar
- ☆ Problemadäquate Architektur

1.4 Produkte

- ☆ Resultat des Architekturentwurfs: Software-Architektur
  - ➪ Niedergelegt in einem Dokument (genannt Lösungskonzept, Software-Architektur, Architectural design [document], o.ä.)
  - ➪ Findet ihren Niederschlag in Coderahmen und/oder Code
  - ➪ Die Struktur des Codes ist eine bruchfreie oder zumindest formal rückverfolgbare Umsetzung der Architektur
  - ➪ Dokumentiert auch Einbettung in vorhandene Software sowie Beschaffungs- und Wiederverwendungsentscheide
- ☆ Resultat des Detailentwurfs: Wohldokumentierter Coderahmen für jede Komponente des Codes
  - ➪ Die Struktur des Detailentwurfs ist eine möglichst bruchfreie, mindestens aber formal rückverfolgbare Umsetzung der Architektur
  - ➪ Die Struktur des Codes ist eine 1:1-Umsetzung der Struktur des Detailentwurfs

1.5 Der Entwurfsprozess

1.5.1 Die Hauptaufgaben des Architekturentwurfs

Aufgabe analysieren

- Anforderungen verstehen
- Vorhandene bzw. beschaffbare Technologien und Mittel analysieren

Architektur modellieren und dokumentieren

- **Grundlegende Systemarchitektur festlegen**
  - Muster — Kapitel 3, 6
  - Metaphern — Kapitel 6 und Vorlesung «Modellierung», Kapitel 7
  - ⇨ Stil — Kapitel 1.7

- **Modularisieren**
  - Gliederung der zu erstellenden Software in Komponenten
  - Abgrenzung der Module
  - Festlegung von Verantwortlichkeiten und Entwurfsgeheimnissen
  - Definition der Schnittstellen
  - Wer kommuniziert was mit wem
  - Wie wird kommuniziert
  - Verträge

- **Nebenläufige Lösungen in Prozesse gliedern**
  - Parallele/Zeitlich verzahnte Ausführung von Aktivitäten analysieren
  - Festlegung der Prozesse
  - Zuordnung von Modulen zu Prozessen
  - Festlegung der Kommunikationsmittel; Zuordnung von Zusammenarbeit zu Mitteln

- **Ressourcen zuordnen**
  - Module -> Prozesse
  - Prozesse und Daten -> Prozessoren, Speicher
  - Zusammenarbeit -> Kommunikationsmittel, Netze

- **Teilkonzepte für Querschnittsaufgaben erstellen**

  Erstellung aspektorientierter Konzepte, zum Beispiel

  - Konzeptionelles Datenbankschema
  - Mensch-Maschine-Kommunikationskonzept
  - Fehlerbehandlungs-, Fehlertoleranz-, Sicherheitskonzepte

- **Lösungskonzept (als Dokument) erstellen**

  Aufbau:
  - Einleitung
  - Struktur der Lösung
  - Aspektbezogene Teilkonzepte
  - Voraussetzungen und benötigte Hilfsmittel

Lösungskonzept prüfen

- Anforderungen/Kundenwünsche erfüllt?
- Softwaretechnisch gut?
- Wirtschaftlich?

1.5.2 Hauptaufgaben des Detailentwurfs

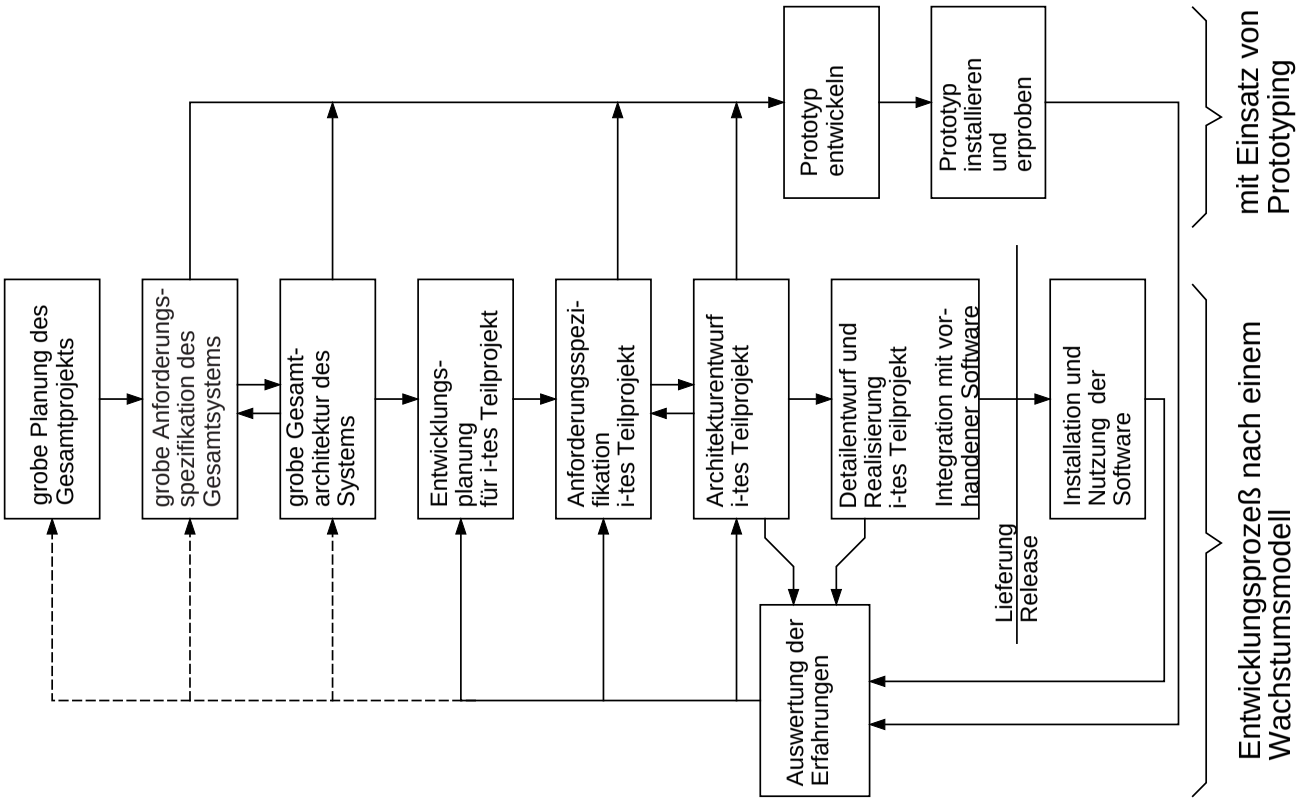
- Abbildung der Module und Prozesse auf die verfügbaren Konstrukte der verwendeten Programmiersprache(n)
- Erstellung von Coderahmen und Implementierungsskizzen für alle Module und Prozesse
- Detaillierte Ausarbeitung aller Aspektkonzepte
- Wo noch nicht geschehen: Umsetzung der Aspektkonzepte in den entworfenen Modulen und Prozessen
- ➪ Kann bei Verwendung leistungsfähiger Programmiersprachen und bei Komponenten mit geringen Risiken mit der Codierung zusammenfallen

1.5.3 Einbettung

- Erster Schritt der Lösung
- Wenn Anforderungsspezifikation vorliegt
- Vorgabe für Codierung

Aber

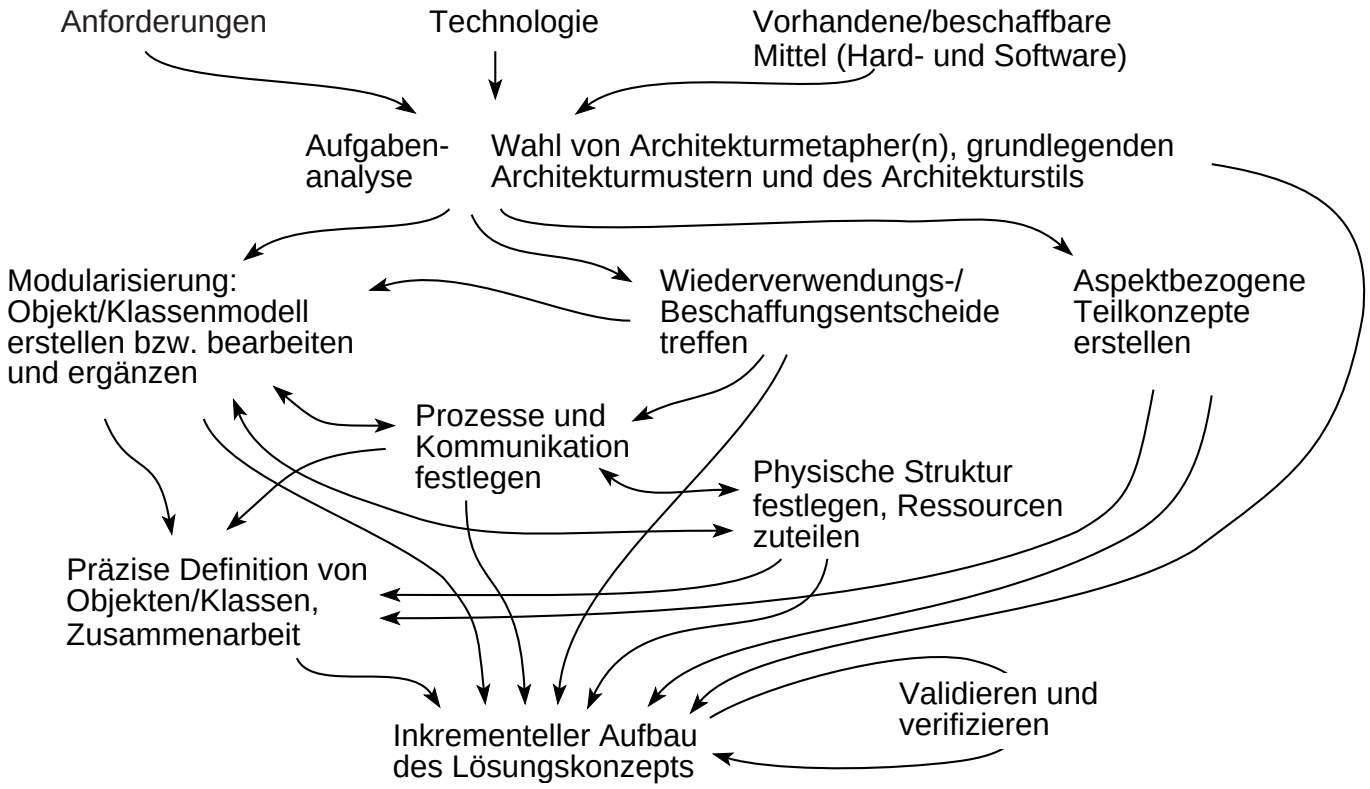
- Hierarchische Verzahnung von Anforderungen und Lösungen
- Zeitliche Verzahnung von Anforderungen, Entwürfen und Code bei evolutionären Prozessen



1.5.4 Vorgehen

- Keine Patentrezepte oder algorithmischen Wege
- Vorgehen ist abhängig vom verwendeten Entwurfsstil und vom gewählten Entwicklungsmodell
- Nachfolgend: Mögliches Vorgehen bei objektorientiertem, evolutionärem Entwurf

Vorgehen und Zusammenhänge:



1.5.5 Variantenbehandlung

- Erkennen
- Beurteilen: Kostengünstigste Variante bestimmen:
  - Kosten der Variante (einschließlich Folgekosten!)
  - Kosten der Untersuchung (!)
  - Je größer / teuerer der Untersuchungsgegenstand, desto aufwendiger darf die Untersuchung sein
- Entscheiden

1.5.6 Beschaffung und Wiederverwendung

- Ist ein Konzeptentscheid
- Für jede Komponente untersuchen, ob die Option Beschaffung bzw. Wiederverwendung besteht
- Falls ja, Beschaffung / Wiederverwendung vs. Eigenentwicklung als Lösungsvarianten untersuchen und entscheiden

1.6 Das Lösungskonzept

- Dokumentiert das Ergebnis des Architekturentwurfs
- Möglicher Aufbau:

---

1. Einleitung

1.1 Überblick

Überblick über die gewählte Lösung

1.2 Ziele und Vorgaben

Beschreibung von Entwurfszielen und Vorgaben, die nicht in der Anforderungsspezifikation stehen

1.3 Einbettung und Abgrenzung

- Wo und wie ist das konzipierte System eingebettet
- Wie und über welche Schnittstellen wird mit der Umwelt kommuniziert

1.4 Lösungsalternativen

Betrachtete, aber schließlich verworfene Lösungsalternativen: Kurze Skizze jeder Alternative, Grund für die Verwerfung

2. Struktur der Lösung

2.1 Übersicht

- Architekturstil, Metapher(n) und Architekturmuster, die der Architektur zugrunde liegen
- Teilsysteme und ihre Aufgaben

2.2 Prozessstruktur

Prozesse und Kommunikation zwischen den Prozessen

2.3 Modulare Struktur

Module und ihre Zusammenhänge, bei objektorientiertem Entwurf Klassen- bzw. Objektmodelle

2.4 Entwurf der Module

- Beschreibung der Schnittstellen
- ggf. Hinweise zur geplanten Implementierung

2.5 Physische Struktur

- Physische Gliederung der Software in Pakete, Komponenten, etc.
- Ressourcenzuordnung

3. Aspektbezogene Teilkonzepte

Ein Unterkapitel je interessierendem Aspekt, zum Beispiel Datenhaltungskonzept, Mensch-Maschine-Kommunikationskonzept, Fehlerbehandlungskonzept, Fehlertoleranzkonzept, Sicherheitskonzept, etc.

4. Voraussetzungen und benötigte Hilfsmittel

4.1 Benötigte Software

Beschreibung der benötigten (fertigen) Software, welche für Entwicklung und/oder Betrieb des Systems zu beschaffen bzw. zu verwenden ist

4.2 Benötigte Hardware

Beschreibung der benötigten Hardware, welche für Entwicklung und/oder Betrieb des Systems zu beschaffen bzw. zu verwenden ist

4.3 Benötigtes Umfeld

Charakterisierung der für den Betrieb des Systems erforderlichen organisatorischen und /oder technischen Strukturen und Abläufe

Quellennachweis

1.7 Architekturstile

Architekturstil – Leitlinien für die Gestaltung der Architektur

- ☆ Verwendete Modularten
- ☆ Verwendete Arten von Kooperation zwischen den Modulen
- ☆ Benutzung passender Architekturmuster
- ☆ Orientierung an einer Architekturmetapher

➜ Architekturstil

1.7.1 Funktionsorientierte Architektur (Structured Design)

- ❍ Jeder Modul berechnet eine Funktion
- ❍ Zur Realisierung einer Funktion können andere, einfachere Funktionen aufgerufen werden
- ❍ Daten werden über Parameter oder gemeinsame Speicherbereiche ausgetauscht
- ➥ Entwurf = Hierarchie aufeinander aufbauender Funktionen
- ❍ Meist nach dem Prinzip Eingabe – Verarbeitung – Ausgabe strukturiert
- ❍ Typischer Vertreter: "Structured Design"
- ❍ Problem: Daten und funktionsübergreifende Probleme können nicht gekapselt werden -> Ungenügende Abstraktion -> Verständnis und Änderbarkeit erschwert
- ➥ Liefert in vielen Fällen keine gute Modularisierung

1.7.2 Datenorientierte Architektur (Datenabstraktionen)

- Modularisierung nach dem Geheimnisprinzip
- Datenstruktur und alle darauf möglichen Operationen sind zusammengefasst: *Datenabstraktion*
  - notwendig zum Verbergen von Entwurfsentscheidungen
  - Realisierung durch Abstrakte Datentypen
- System besteht aus Menge aufeinander aufbauender Datenabstraktionen
  - Jeder Modul bietet Leistungen an
  - Module benutzen die Leistungen anderer Module zur Realisierung der eigenen Leistungen
  - ➥ Benutzungshierarchie
  - Regelung der Zusammenarbeit durch Verträge (Design by Contract)
- Zusätzlich häufig Gliederung in Schichten
- Stil besonders geeignet zur Realisierung von Information Hiding
- Gute Abstraktion -> Entwürfe sind leicht verstehbar und änderbar

1.7.3 Objektorientierte Architektur (-> Kapitel 2)

- Modul repräsentiert Objekt des Problembereichs oder benötigtes Informatik-Element
- System besteht aus Menge kooperierender Objekte
- Abstrakte Beschreibung gleichartiger Objekte ➔ Klasse
- Geeignet gebildete Klassen beachten das Geheimnisprinzip ➔ gute Modularisierung
- Systematischer Zusammenhang zwischen allgemeinen und speziellen Objekten: Objekte von Spezialklassen *erben* alle Strukturen und Operationen der übergeordneten allgemeinen Klassen
  - ➔ Spezialisierungs- (Generalisierungs-) Hierarchie
    - ermöglicht problemnahe Modellierung
    - schränkt jedoch Anwendbarkeit des Geheimnisprinzips ein
- Durch Vererbung massiv erhöhte Flexibilität
  - \+ Software ist erweiterbar
  - – bei unsachgemäßer Verwendung starke Nebenwirkungen auf Verstehbarkeit und Änderbarkeit

1.7.4 Prozessorientierte Architektur (-> Kapitel 4)

- System besteht aus Menge unabhängig arbeitender, untereinander kooperierender Akteure
- Akteure sind typisch als Prozesse realisiert
- Prozesse kooperieren durch Austausch von Nachrichten oder durch Zugriff auf gemeinsame Speicherbereiche
- Prozesse sind die Module der obersten Stufe
- Jeder Prozess ist typisch ein sequentiell ablaufender Systemteil
- Prozesse sind selbst wieder modularisiert, z.B. in objektorientiertem Stil

1.7.5 Komponentenorientierte Architektur (-> Kapitel 5)

- Weiterentwicklung objektorientierter Architekturen

Komponente (im engeren Sinn) – Stark gekapselte Menge zusammengehöriger Objekte / KIassen, die eine gemeinsame Aufgabe lösen

- System besteht aus einer Menge von (möglicherweise geographisch verteilten) Komponenten, die über einen Makler kommunizieren
- Komponenten kennen
  - Schnittstellen ihrer Partnerkomponenten
- ... kennen nicht
  - Art der Realisierung der Kommunikation
  - Geografische Lokalisierung
  - Implementierung der Partnerkomponenten

1.7.6 Datenflussorientierte Architektur

- Die Software wird in Aktivitäten und Datenflüsse (sowie ggf. Speicher) gegliedert
- Aktivitäten arbeiten, wenn die von ihnen benötigten Daten vorliegen

  -> Systemsteuerung durch Datenfluss

Typische Vertreter

- Leitung-Filter-Architekturen, z.B. UNIX pipes
- Regler
- Strukturierte Analyse