Vorlesung "Softwaretechnik"
Entwurf: Architektur

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

- Archit. = Gesamtstruktur
  - nichtfunkt./funkt. Anforderngn.
  - globale Eigenschaften
- Wiederverwendbare Architekturen
  - Standard-Architekturen, Referenz-Architekturen
- Architekturen selber finden
  - Architekturstile
- Modularisierung
  - Modulbegriff
  - Kriterien für Aufteilung

Abbildungen z.T. aus einer Lehreinheit von Gregor Engels

Wo sind wir gerade?: Taxonomie "Die Welt der Softwaretechnik"

Freie Universität Berlin

Welt der Problemstellungen:

- Produkt (Komplexitätsprob.)
  - Anforderungen (Problemraum)
  - **Entwurf (Lösungsraum)**
- Prozess (psycho-soziale P.)
  - Kognitive Beschränkungen
  - Mängel der Urteilskraft
  - Kommunikation, Koordination
  - Gruppendynamik
  - Verborgene Ziele
  - Fehler

Welt der Lösungsansätze:

- Technische Ansätze ("hart")
  - **Abstraktion**
  - **Wiederverwendung**
  - Automatisierung
- Methodische Ansätze ("weich")
  - Anforderungsermittlung
  - **Entwurf**
  - Qualitätssicherung
  - Projektmanagement

Wo sind wir?: Entwurf

Freie Universität Berlin

- Einsicht: Man sollte *vor* dem Kodieren über eine günstige Struktur der Software nachdenken
  - und diese als Koordinationsgrundlage schriftlich festhalten

- Prinzipien:
  - **Trennung von Belangen**
  - ***Architektur***: Globale Struktur festlegen (Grobentwurf), insbes. für das Erreichen der nichtfunktionalen Anforderungen
  - **Modularisierung**: Trennung von Belangen durch Modularisierung, Kombination der Teile durch Schnittstellen (information hiding, Lokalität)
  - ***Wiederverwendung***: Erfinde Architekturen und Entwurfsmuster nicht immer wieder neu
  - **Dokumentation**: Halte sowohl Schnittstellen als auch zu Grunde liegende Entwurfsentscheidungen und deren Begründungen fest

Wo sind wir? (2)

Wo steht man nach der Anforderungsermittlung?

- Man weiß (mehr oder weniger) **was** man bauen will. D.h.:
- Man kennt die funktionalen Anforderungen
  - z.B. beschrieben durch Use Cases etc.
  - und konkretisiert durch das Analysemodell
- Man kennt die nichtfunktionalen Anforderungen
  - wenige oder viele, lasch oder stringent
  - betreffend z.B. Effizienz, Bedienarten, Bedienbarkeit, Robustheit, Sicherheit, Schutz, Erweiterbarkeit, Technologievorgaben, Zuverlässigkeit, Verfügbarkeit, Erlernbarkeit u.a.m.

Wo sind wir? (3)

Freie Universität Berlin

Was müssen wir als nächstes leisten?

- Herausfinden **wie** man es bauen kann/sollte.
  D.h.:
- Entscheiden wie/wo/wodurch die funktionalen Anforderungen (Funktionen) umgesetzt werden
  - z.B. Technologieauswahl, Modulzerlegung, Wiederverwendung
  - Wir klammern Technologieauswahl und Wiederverwendung aus
  - Frage 1 ist also: **Wie zerlegt man ein System klug in Teile?**
- Herausfinden, wie man dabei die nichtfunktionalen Anforderungen alle erfüllen kann
  - nichtfunktionale Anforderungen haben meist globalen Charakter, werden also nicht von nur wenigen Modulen realisiert
  - Frage 2 ist also: **Wie findet man eine Gesamtstruktur mit den gewünschten globalen Eigenschaften?**

Suche nach einer Gesamtstruktur

Freie Universität Berlin

- Das Problem bei der Suche nach einer geeigneten Gesamtstruktur ist folgendes:
  - Die globalen Eigenschaften sind meist emergente Eigenschaften
    - d.h. sie lassen sich nicht den Einzelteilen zuweisen, sondern entstehen erst aus deren Zusammenwirken
    - z.B. Speicherbedarf, Robustheit, Verfügbarkeit, Sicherheit
  - Emergente Eigenschaften sind sehr schwierig im Voraus abzuschätzen

- Muss ich also mehrere Gesamtstrukturen ausprobieren, bis ich eine passende finde?
  - Meist nicht, denn es gibt Erfahrungswerte
  - Die gewünschten Eigenschaften verschiedener Systeme ähneln einander
  - und gewisse Arten von Gesamtstruktur haben sich bewährt

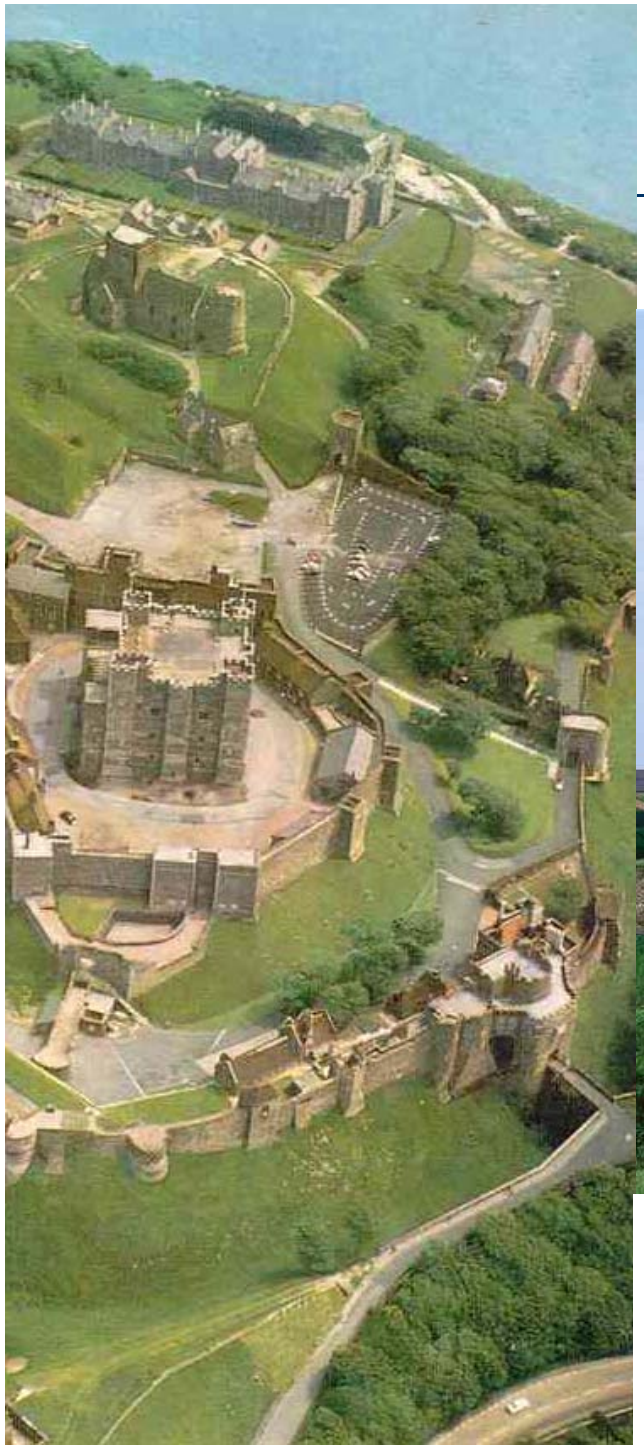
Suche nach einer Gesamtstruktur (2)

Freie Universität Berlin

- Meist gibt es ein paar dominante nichtfunktionale Eigenschaften
- und eine bekanntermaßen dazu gut passende Gesamtstruktur

- Dann benutzt man solche bekannten Gesamtstrukturen
  - oft "Architekturen" genannt

- Wenn Sie ein ganz ungewöhnliches, neuartiges System bauen wollen, haben Sie es allerdings evtl. schwer!
  - Deshalb lohnt es sich, Neuartigkeit zu begrenzen (wo das geht)
  - Siehe normales vs. radikales Vorgehen

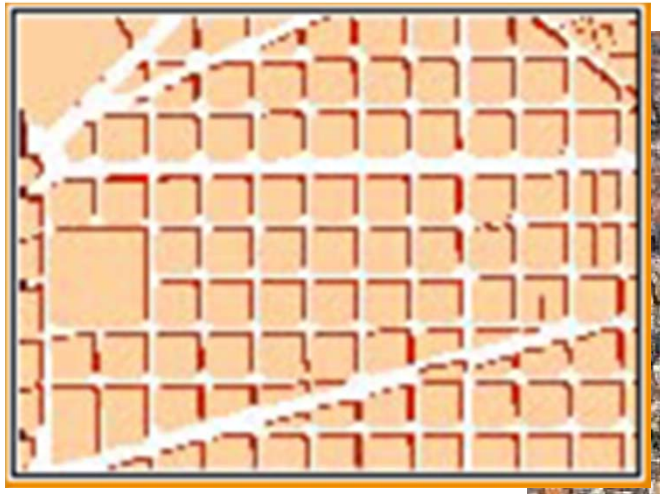


Architekturziel: Verteidigen

Teilziele z.B. Angreifer aufwärts kämpfen lassen und immer möglichst von 3 Seiten beschießen können

Lösungsansätze:
Mauern mit auskragenden Türmen;
mehrere konzentrische Bastionen;
innere Bastionen überblicken äußere

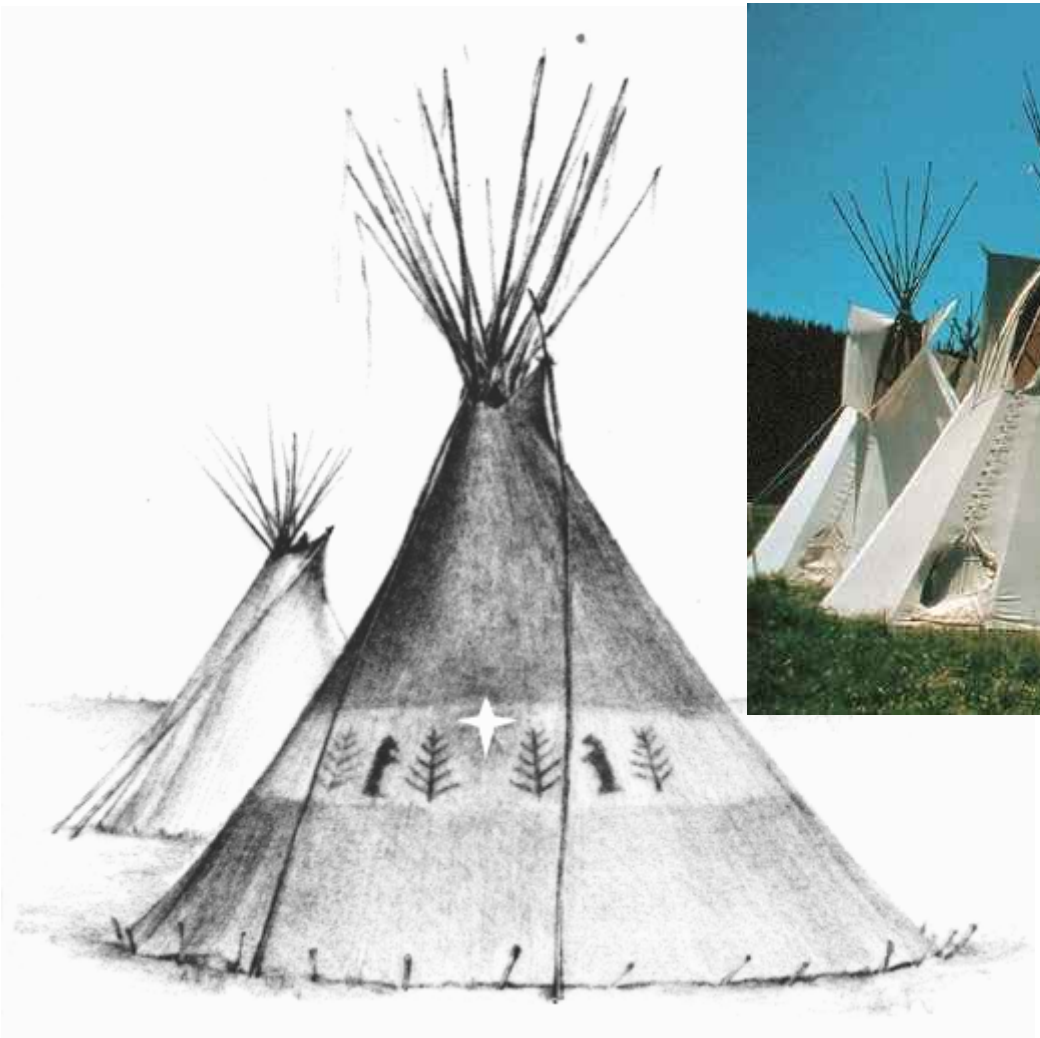
Architekturziel: Verkehrsfluss



Architekturziel: Erweiterbarkeit

Universität Paderborn

Architekturziel: Mobilität



Architekturziel: Beeindrucken

Un-gewöhnliches Ziel ➔ Keine Standardlösung

Achtung: Das waren nur die Hauptziele.
Jede Architektur hat mehr als ein Ziel.

Swiss Re building, London
"The Gurken"

Definition Softwarearchitektur

Freie Universität Berlin

- *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them*
  - Bass, Clements, Kazman:
    "Software Architecture in Practice" (2nd edition),
    Addison-Wesley 2003
  - Diese und Dutzende weiterer Definitionen siehe
    http://www.sei.cmu.edu/architecture/definitions.html

- Diese Definition betont also die Module ("elements"), deren Schnittstellen ("visible properties") und die Art, wie diese miteinander in Beziehungen stehen ("relationships")

Definition Softwarearchitektur (2)

Freie Universität Berlin

- Andere Definitionen betonen z.B.:
  - die Entscheidungen, die man zur Bildung der Struktur getroffen hat (inkl. Auflösen von Konflikten wie Effizienz ←→ Wartbarkeit)
  - die Erfüllung der funktionalen und nichtfunktionalen Anforderungen, die damit erreicht wird
  - die Wiederverwendung (oder mögliche Wiederverw.) von Teilen
  - das dynamische Verhalten des Systems
  - u.a.
- Es gibt also keine einheitliche Definition, was eine Architektur ausmacht
- Aber es gibt eine Gemeinsamkeit aller Definitionen:
  Es geht stets um eine globale Betrachtung!
  - Die aber, wenn man sie gründlich macht, bis auf die Detailebene herunterreicht
  - Deshalb gibt es z.B. so viele UML-Diagrammarten
    - → verschiedene Sichten und Detailniveaus

Beispiele für standardisierte Ideen für SW-Architekturen

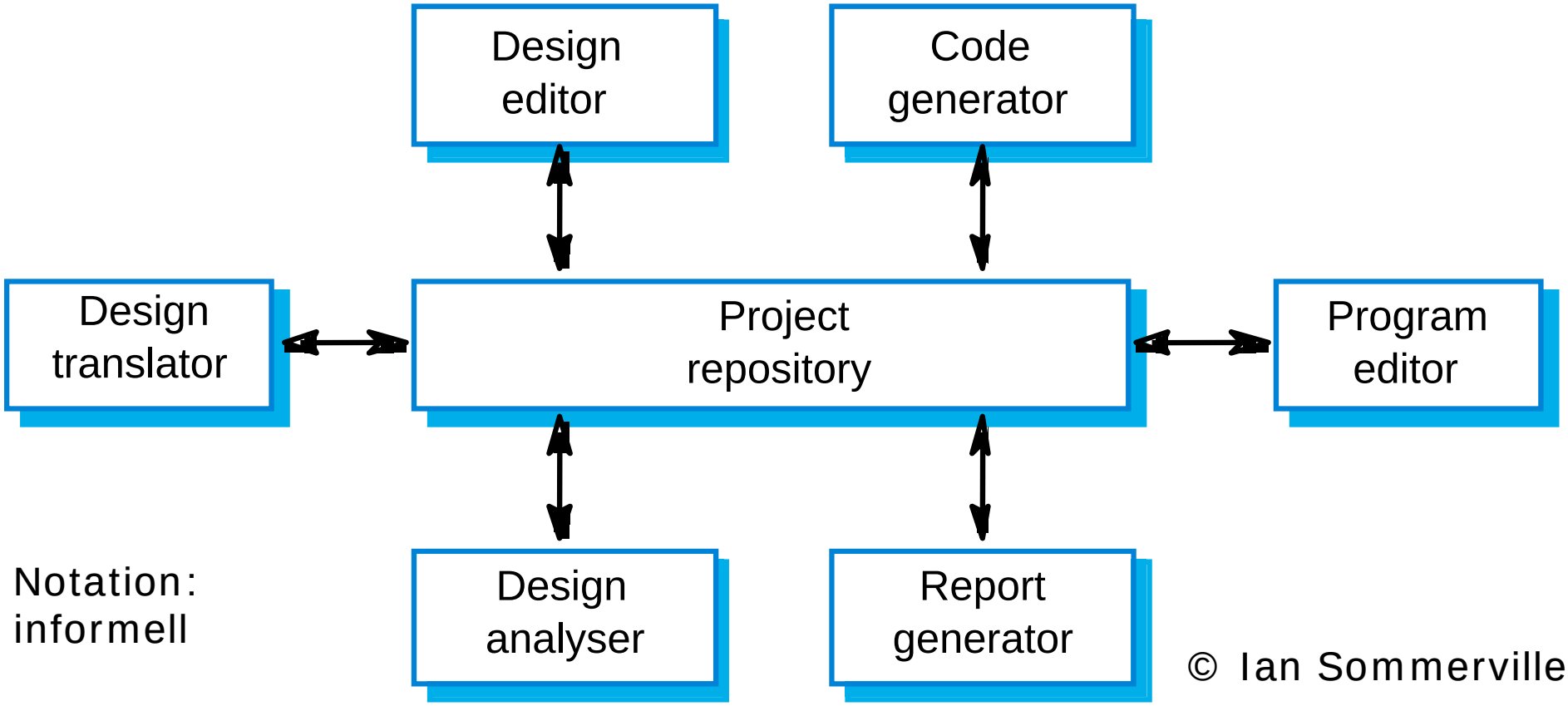
Freie Universität Berlin

- Es gibt für eine Reihe wiederkehrender Mengen nichtfunktionaler Anforderungen etablierte SW-Architekturen oder zumindest Architekturstile
  - Klient-/Dienstgeber-Architektur (client/server arch.)
    - eine einfache Sorte verteilter Architekturen
  - Ereignisgesteuertes System (event-based arch.)
    - eine Sorte lose gekoppelter Architekturen
  - Ablage-basierte Architektur (repository arch.)
    - noch eine Sorte lose gek. A.; oft mit Ereignissteuerung verbunden
  - Unterbrechungsorientiertes System (interrupt-based system)
    - eine Architektur für kleinere Echtzeitsysteme
  - Mehrschicht-Architektur (layered arch.)
    - ein allgemeiner A.stil, der mit vielen anderen A.ideen verbunden werden kann
  - *JavaEE*-Architektur, *CORBA*-Architektur, *.NET*-Architektur
    - technologiezentrierte Architekturen

Beispiel: Ablagearchitektur für CASE-Werkzeug

Freie Universität Berlin

- Idee: Einzelne Werkzeuge kommunizieren (fast) nur indirekt über die Datenablage, kaum direkt



Notation: informell

© Ian Sommerville

Warnung:
Das Bild ist nicht die Architektur!

Freie Universität Berlin

- **Achtung, ganz wichtig!**:
  Die Kästchen und Pfeile auf obigem Bild
  sind nicht "die Architektur"!

Sondern:

1. Sie sind nur Repräsentationen
   - (die auch ganz anders aussehen könnten)
2. Sie repräsentieren nur sehr wenige Aspekte der Architektur
   - hier: die wichtigsten Kommunikationsbeziehungen der Subsysteme
   - viele andere Aspekte bleiben völlig ungenannt (z.B. Steuerung)
3. Selbst diese Aspekte sind nur sehr ungenau wiedergegeben
   - hier z.B.: Wie oft wird kommuniziert? Welche Datenstrukturen? Welche Kommunikationsmechanismen? Wie ausgelöst? etc.

Architektur und Architektursichten

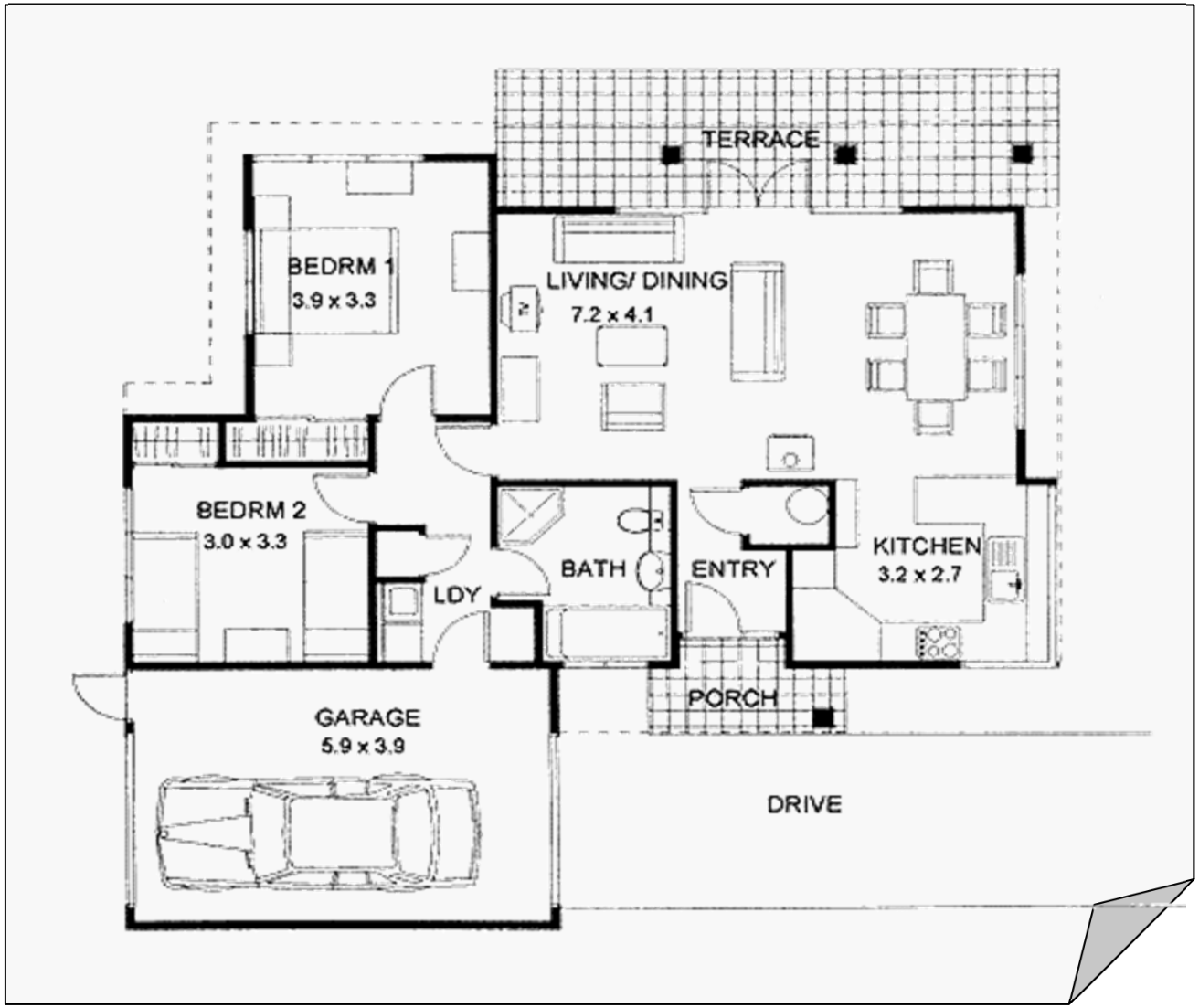
Freie Universität Berlin

Eine vollständige Architekturbeschreibung ist ein komplexes Modell und besteht aus viel mehr als einem Bild:

1. Man benutzt viele Bilder anstatt nur eines
   - verschiedene Arten von Bild für verschiedene Aspekte
     - z.B. verschiedene UML-Diagrammarten
   - verschiedene Bilder für verschiedene Teilsysteme (auf den feineren Detailniveaus)
     - z.B. viele UML-Teil-Klassendiagramme für das statische Modell
2. Neben den Bildern braucht man zusätzliche Information
   - meist textueller Natur, z.B. Schnittstellendefinitionen
     - z.B. in OCL (Object Constraint Language) oder natürlichsprachlichem Text

Siehe Einheit über Modellierung

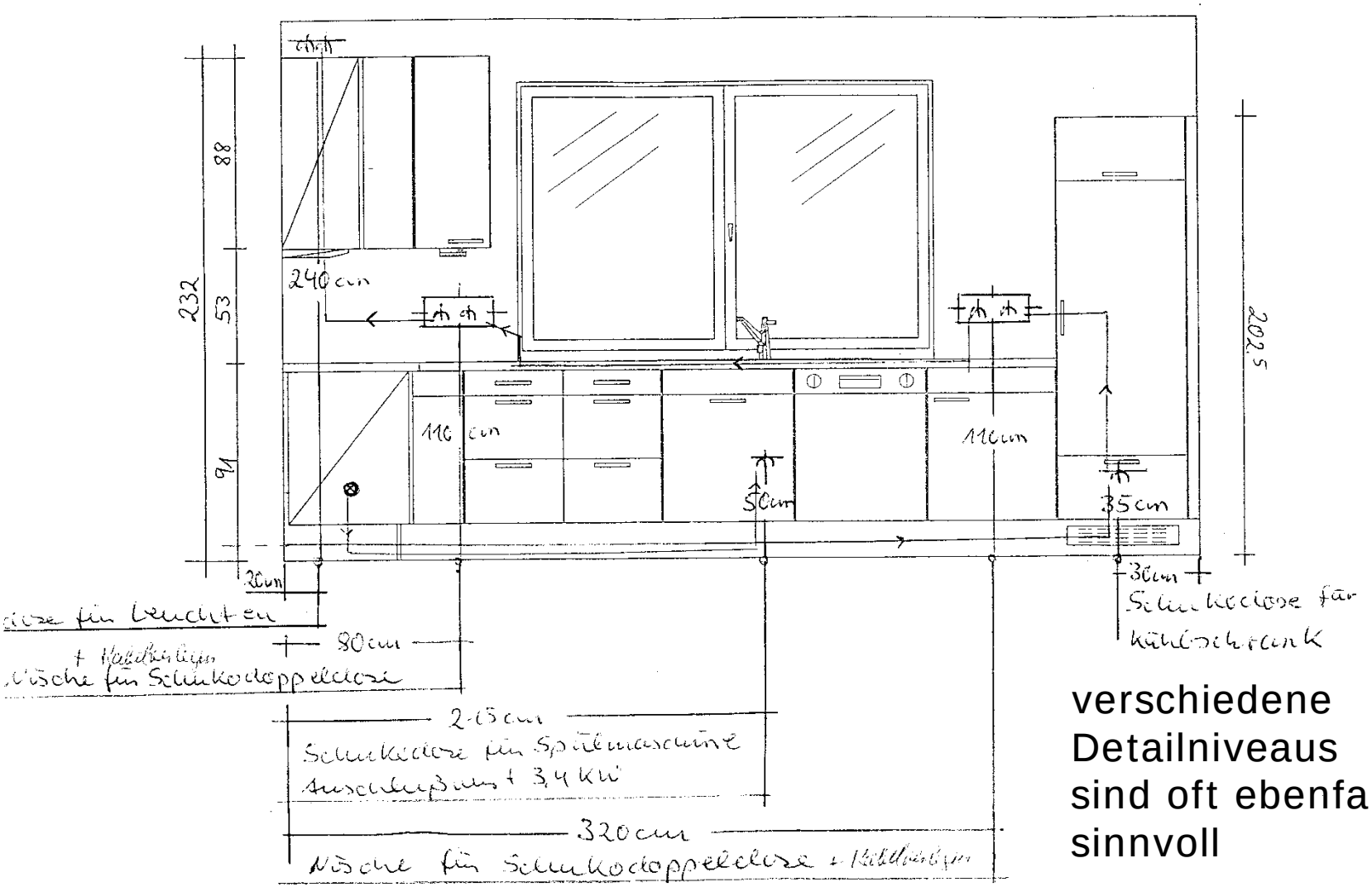
Architektur: Grundriss



Architektur: Aufriss

- Grundriss und Aufriss könnten zwei Sichten desselben Hauses sein
  - (sind sie hier allerdings nicht)

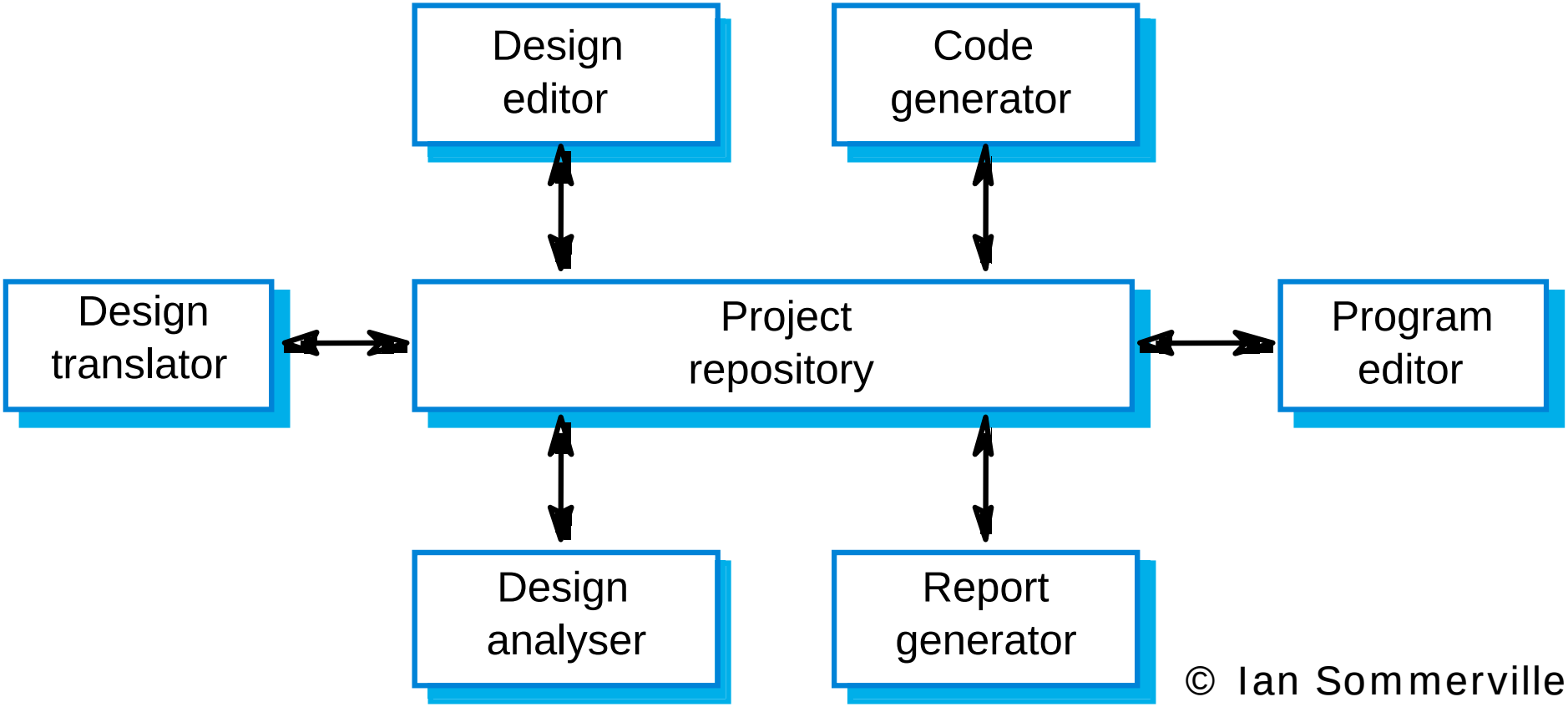
Detailansicht: Küchenzeile



verschiedene Detailniveaus sind oft ebenfalls sinnvoll

Nochmal: Ablagearchitektur für CASE-Werkzeug

- Idee: Einzelne Werkzeuge kommunizieren (fast) nur indirekt über die Datenablage, kaum direkt



© Ian Sommerville

Beispiel (2): Vor- und Nachteile von Ablage-basierten Architekturen

Freie Universität Berlin

Vorteile:

- Effizenter Austausch großer Datenmengen
- Subsysteme müssen Herkunft der Daten nicht kennen
- Zentrale Datenverwaltung (Rechte, Backup, ...)
- Beschreibung der Daten durch Schema (Meta-Daten)

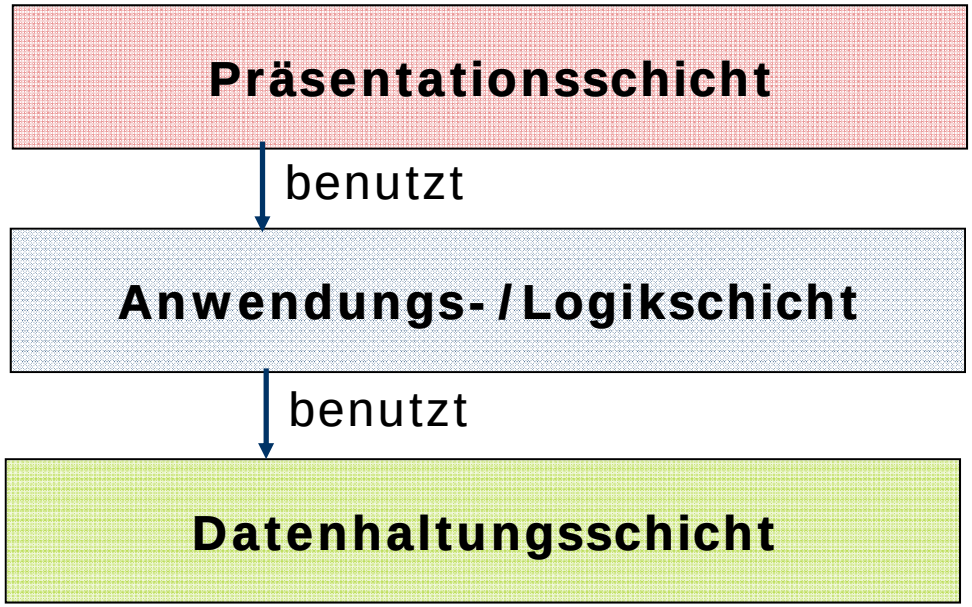
Nachteile:

- Schema begrenzt Flexibilität
- Schemaänderungen sind nur schwer möglich
  - Zu viele Subsysteme sind betroffen
- Alle Subsysteme sind an gleiche Datenverwaltungsregeln gebunden
- Ist in verteilten Umgebungen nicht effizient
  - Bewirkt unnötig viel Datentransfer

Beispiel 2: Schichten-Architektur

Freie Universität Berlin

- Ein besonders häufiger Fall:
  Die 3-Schichten-Architektur für Informationssysteme
  - Die Präsentationsschicht kapselt die Interaktionen mit Benutzern oder Systemen
    - Boundary classes
  - Die Logikschicht enthält die Geschäftslogik
    - Control classes
  - Die Datenschicht kümmert sich um die persistente Speicherung aller Daten
    - Entity classes



  - Jedes Modul benutzt nur Module seiner Schicht und der Schichten darunter (und liefert Dienste für die höheren Schichten)
    - Höhere Schichten werden von niederen niemals benutzt

- Notation: informell

Beispiel 2: Schichten-Architektur (2)

Freie Universität Berlin

Vorteile:

- Sehr übersichtliche, aufgeräumte Grobstruktur
- Komplettaustausch ganzer Schichten ist möglich
- Unnötige Kopplungen zwischen Modulen werden vermieden

Nachteile (in anderen Fällen als dem gezeigten):

- Kann umständlich oder unnatürlich sein
  - Manche Abstraktionen sind evtl. nur vorhanden, um die Schichtung herzustellen
- Kann Laufzeit-ineffizient sein
  - wenn man zu viel abstrahiert hat und zu oft "weiterreichen" muss
    - z.B. im ISO/OSI 7-Schichten-Modell

(Für Informationssysteme ist diese Arch. aber meist günstig)

Architekturstile

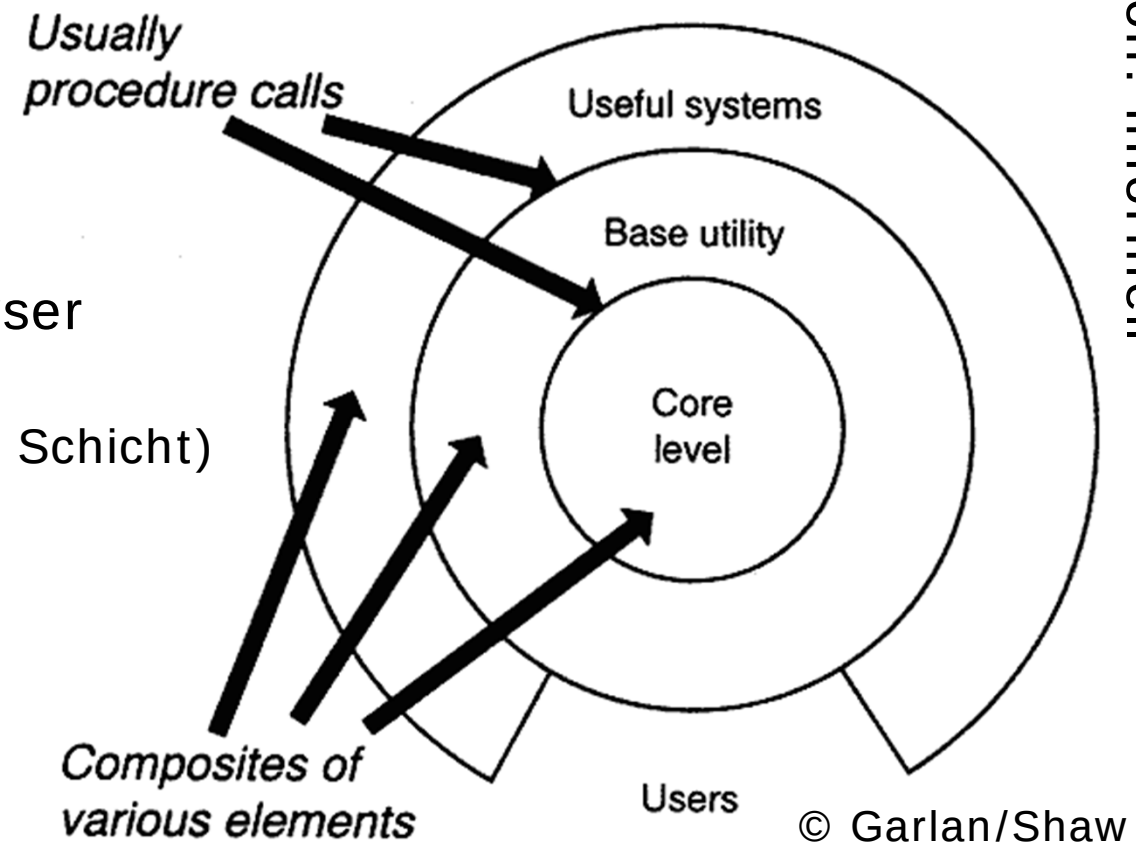
- Die Schichtenarchitektur ist keine komplette Architektur, sondern mehr ein Prinzip oder Stil, wie man ein System (oder Teile davon) organisieren kann

- Es gibt eine kleine Zahl solcher Architekturstile:
  - Schichten
  - Datenflussnetze (Pipes-and-Filters)
  - Objektnetze mit Datenkapseln
  - Ereignissteuerung mit impliziten Aufrufen
  - Ablagebasierte Struktur
  - Interpretierer oder Regelsysteme

- Kurze Beschreibung jedes Stils folgt

Architekturstil Schichten

- Idee: Bilde mehrere Abstraktionsschichten, um
  - von technischen Mechanismen (unterste/innerste Schicht)
  - zu inhaltlichen Lösungen (oberste/äußerste Schicht) zu kommen
- Universell nützlich
- Beispiel: Firefox-Browser (<u>sehr</u> vereinfacht)
  - Linux-Kern (unterste Schicht)
  - libc
  - libxml (u.v.a.m.)
  - Gecko
  - Firefox-Browser (oberste Schicht)



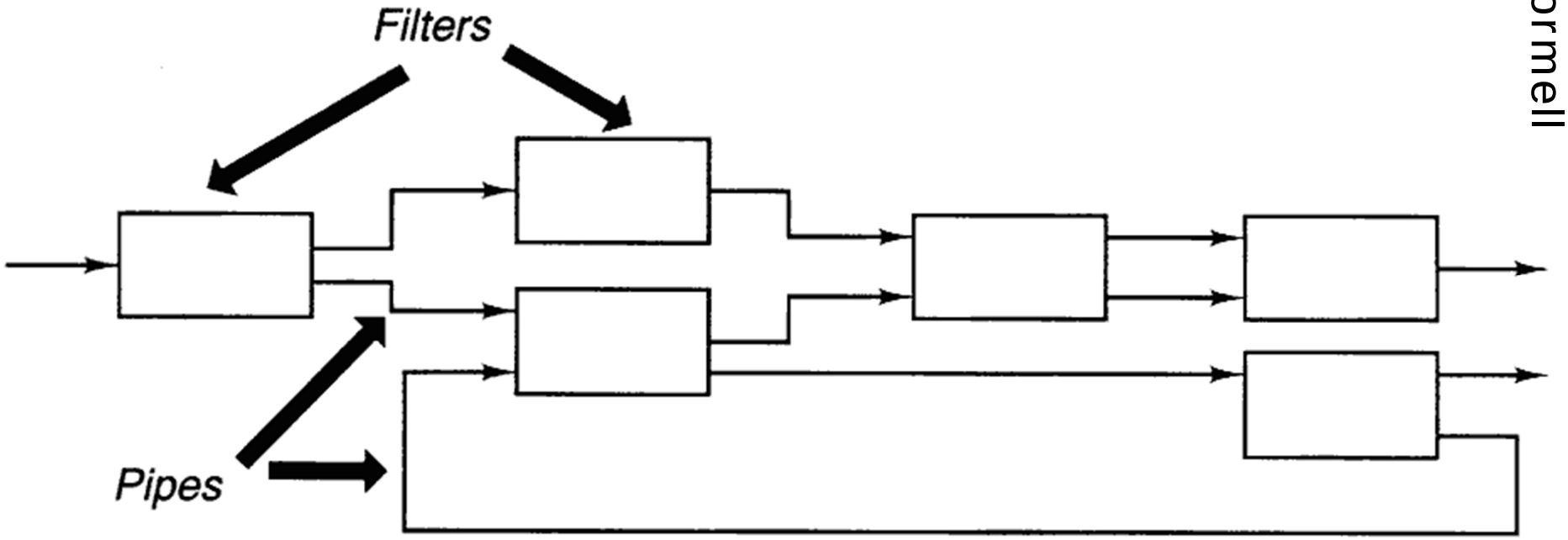
© Garlan/Shaw

- Notation: informell

Architekturstil
Datenflussnetze (Pipes-and-Filters)

Freie Universität Berlin

- Idee: Verknüpfe Verarbeitungsschritte (Filters) nur mittels eines Datenflusses (Pipe)
  - Vorteil: Einfach und klar
  - Nachteil: Für die meisten Zwecke zu unflexibel
    - Ausnahmen gibt es z.B. beim wissenschaftlichen Rechnen (Matrizen)
- Beispiel: Rendering-Pipeline bei Graphikkarten/OpenGL

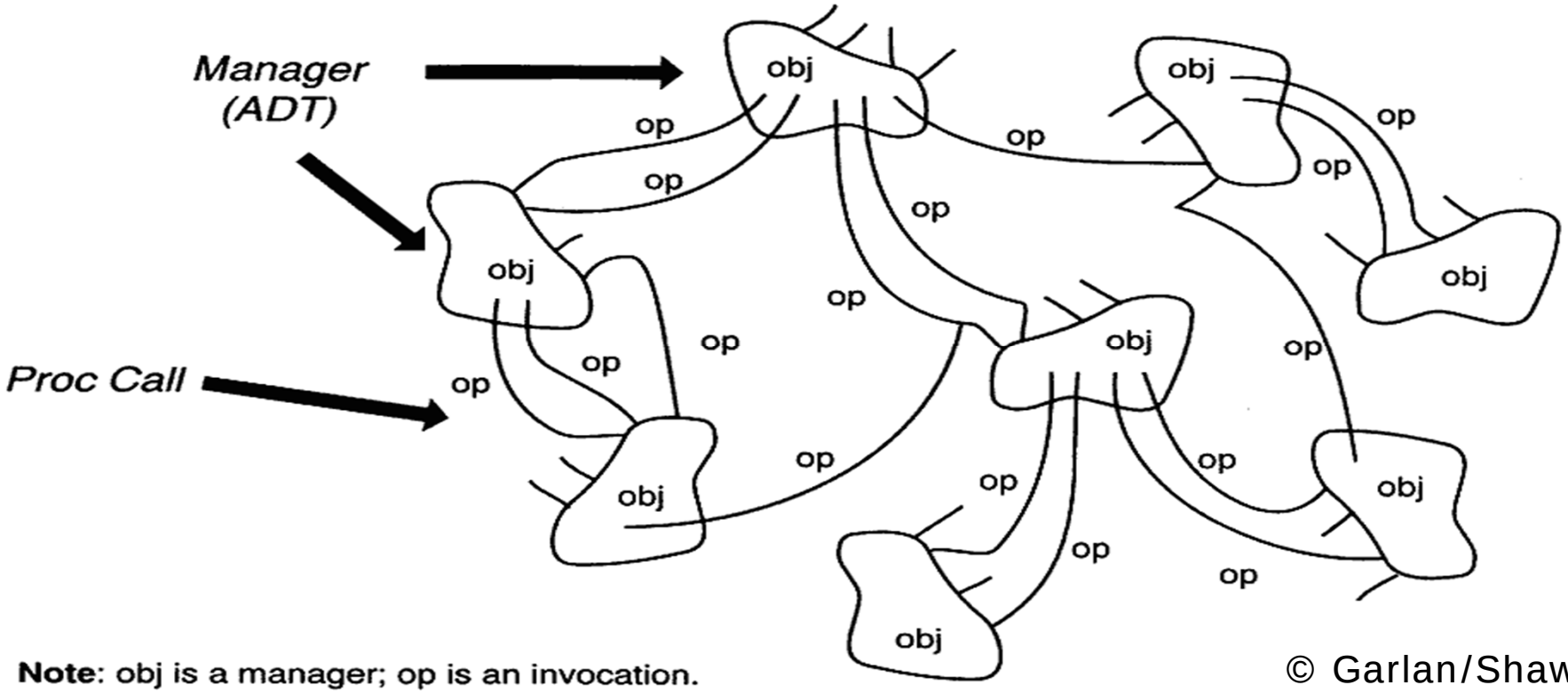


© Garlan/Shaw

- Notation: informell

Architekturstil
Objektnetze mit Datenkapseln

- Idee: Gestalte Objekte und deren Beziehungen komplett problemabhängig (Grundidee der Objektorientierung!)
  - Vorteil: Sehr flexibel
  - Nachteil: Schwer zu erdenken; kann verworren werden



Note: obj is a manager; op is an invocation.

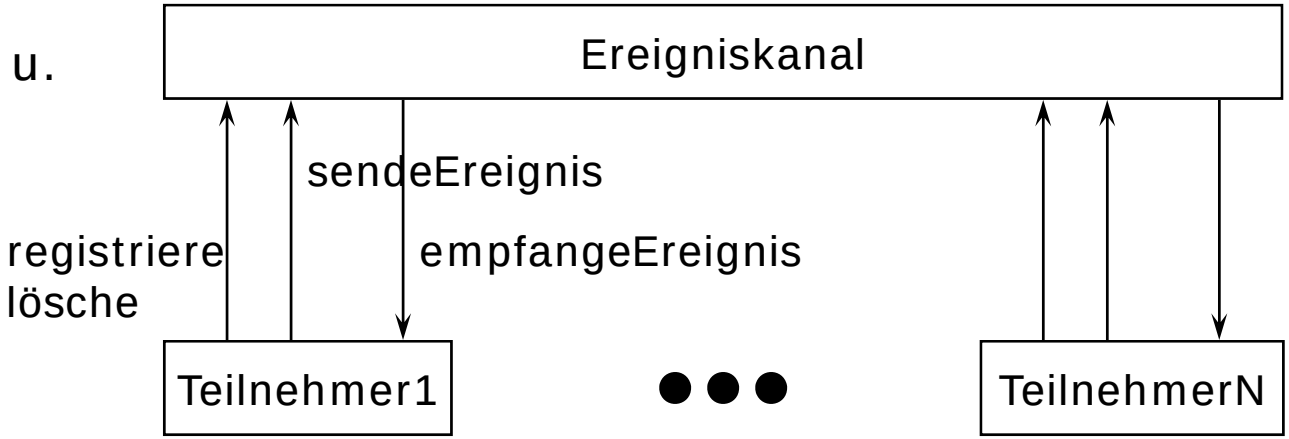
© Garlan/Shaw

- Notation: informell

Architekturstil Ereignissteuerung mit impliziten Aufrufen

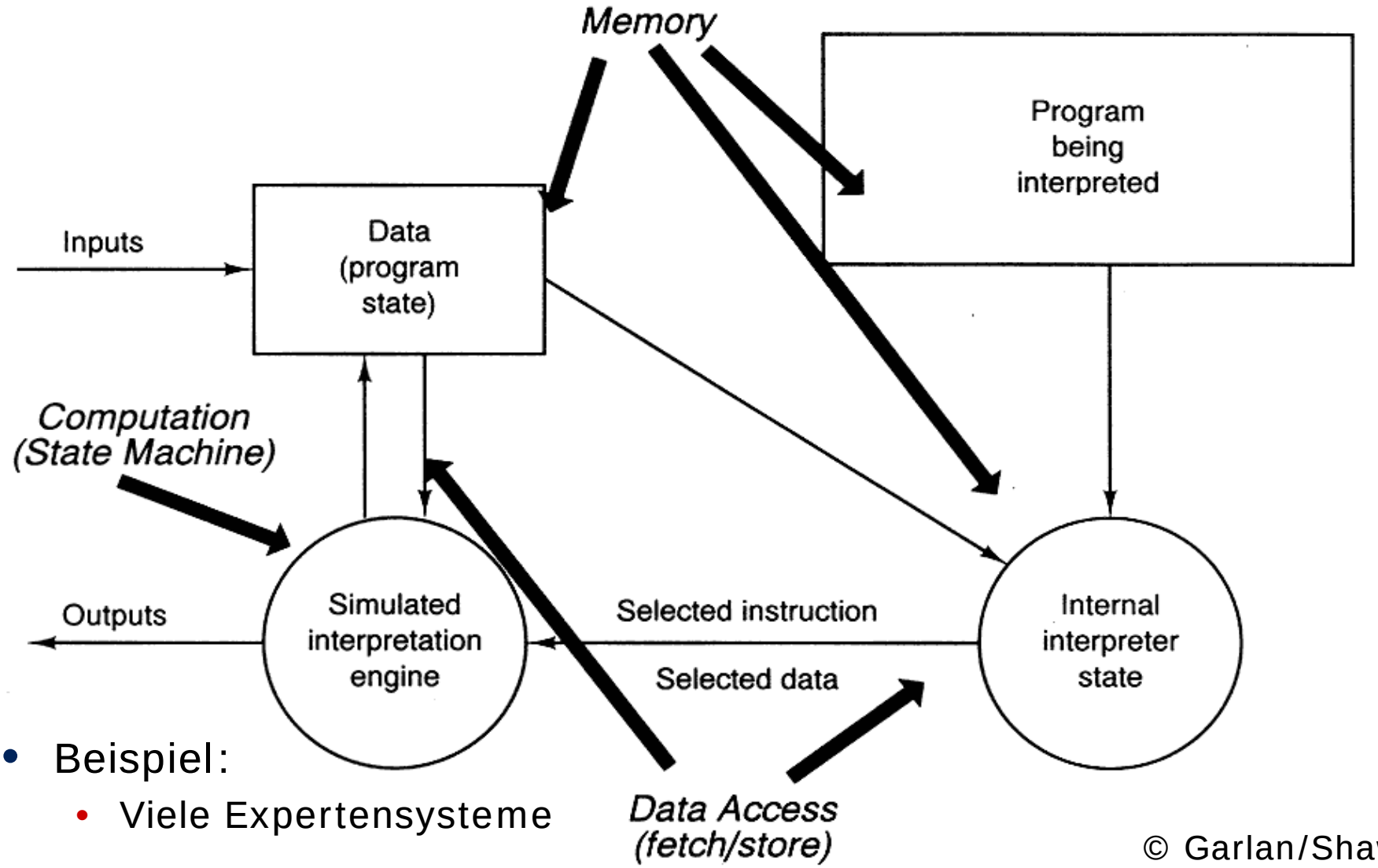
Freie Universität Berlin

- Idee:
  - Verarbeitungseinheiten registrieren sich an zentraler Stelle, um bei bestimmten Ereignissen benachrichtigt zu werden
  - Sie können auch selbst Ereignisse erzeugen
  - Dadurch wird die Systemsteuerung dezentralisiert
- Vorteil:
  - Sehr flexibel und in vielen Fällen änderungsfreundlich
- Nachteil:
  - Verhalten eventuell schwer zu durchschauen u. zu ändern
- Beispiel:
  - fast alle GUI-Frameworks, Eclipse



- Notation: informell

Architekturstil
Interpretierer oder Regelsysteme



Notation: informell

- Beispiel:
  - Viele Expertensysteme

© Garlan/Shaw

Architekturstil Ablagebasierte Struktur (repository, blackboard)

Freie Universität Berlin

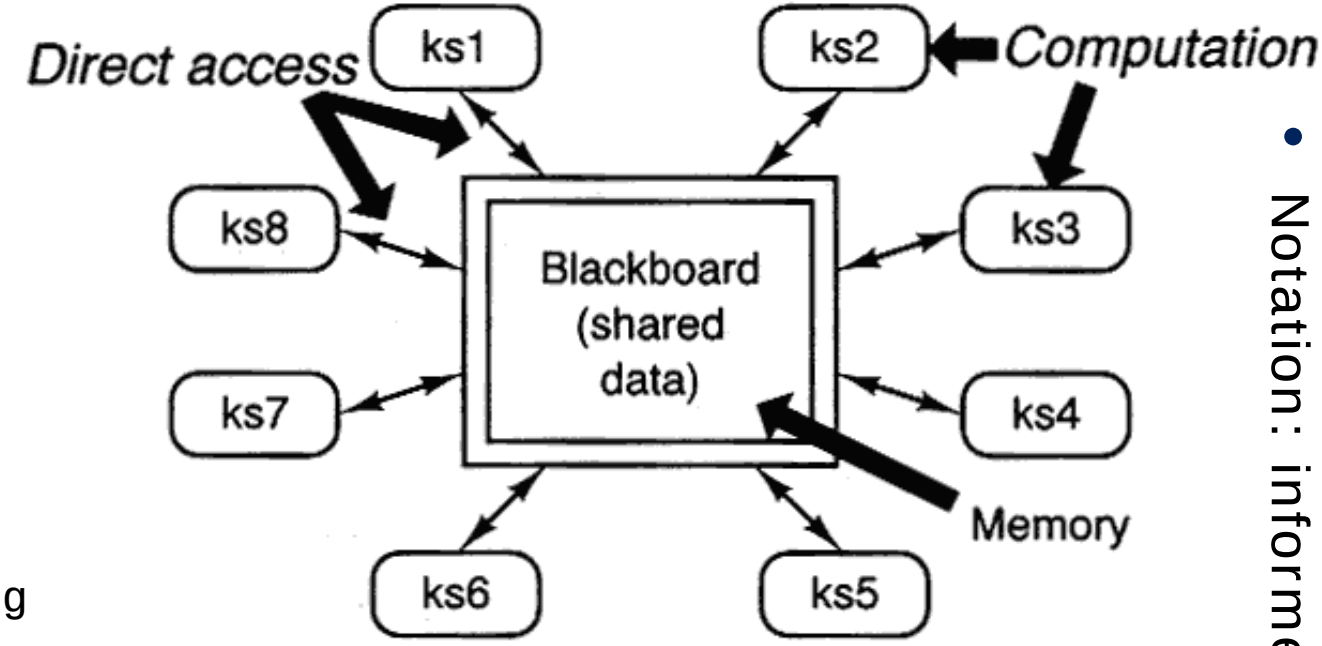
- Idee: Kopple Verarbeitungseinheiten nicht über Kontrollflusssteuerung, sondern nur über gemeinsam genutzte Daten

- Vorteil:
  - Lose Kopplung

- Nachteil:
  - unflexible Steuerung

- Wird oft kombiniert mit Ereignissteuerung

- Beispiele:
  - KI-Software, CASE-Werkzeuge



Note: ks is a knowledge source.

© Garlan/Shaw

- Notation: informell

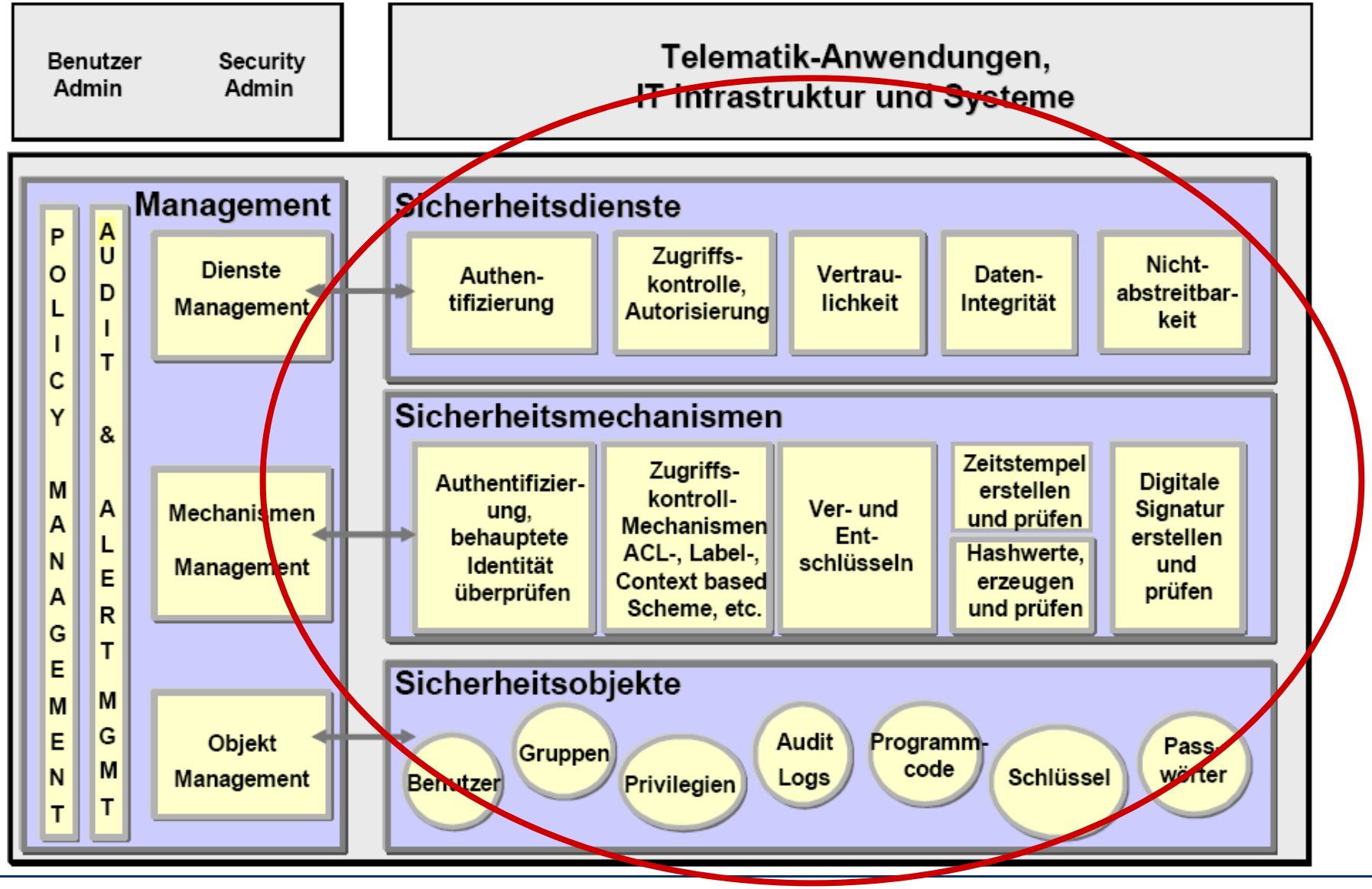
Kombination von Architekturen

Freie Universität Berlin

- Ein großes System enthält insgesamt fast immer mehrere Architekturstile
  - z.B. benutzt fast jedes größere Java-basierte System an mehreren Stellen Schichten, Ereignissteuerung, eine Ablage und natürlich Objektnetze
- Ein großes System kombiniert oft auch mehrere Gesamtarchitekturen miteinander
  - z.B. verschiedene Architekturen in verschiedenen Subsystemen
  - z.B. Kombination einer Verteilungsarchitektur (z.B. Client/Server) mit einer Abhängigkeitsarchitektur (z.B. Schichten)
  - u.s.w.
- Beispiel: Schichten im Aufbau des Sicherheitsdienst-Subsystems der eGK

eGK-Sicherheitsarchitektur: Dienste, Mechanismen, Objekte

Freie Universität Berlin



- Notation: informell

Standard-Architekturen

Freie Universität Berlin

- Es gibt diverse Standard-Architekturen und Referenzarchitekturen auf dem Markt
  - in denen z.T. recht detailliert die Lösungswege für viele wiederkehrende Probleme vorgezeichnet sind
- Eine solche einzusetzen senkt den Entwurfsaufwand enorm
  - weil große Mengen Expertenwissen darin stecken
- Beispiele:
  - QUASAR (Qualitäts-Software-Architektur) von sd&m
    - generische Architektur für betriebliche Informationssysteme
    - formuliert hauptsächlich als Satz von Prinzipien und Methoden
  - JavaEE-Architektur
    - technologie-zentrierte Architektur für Informationssysteme auf Basis der diversen Teiltechnologien der Java Enterprise Edition
  - RM-ODP (Reference Model for Open Distributed Processing)
    - generische Architektur für erweiterbare verteilte Systeme

Genaueres Wissen über Standard-Architekturen

Freie Universität Berlin

- Fast jedes moderne Softwaresystem sollte zumindest weitgehend gemäß einer Standardarchitektur gestaltet sein
  - Normaler Entwurf statt radikaler Entwurf

- Konsequenzen:
  - Befassen Sie sich intensiv mit den Standardarchitekturen in Ihrem späteren Arbeitsbereich
    - Darin steckt sehr viel gutes Know-How ➔ Es gibt viel zu lernen
      - allein 3-Schichtenarchitektur füllt bequem ein Semester
    - Die obigen Darstellungen haben die jeweiligen Beispiele allenfalls angedeutet
  - Seien Sie sehr argwöhnisch gegen Tendenzen, die benutzte Standardarchitektur zu durchbrechen
    - Kann gelegentlich sinnvoll sein,
    - ist aber sehr oft eine *schlechte* Idee:
    - → Genaue Begründung und Abwägung verlangen!

Zwischen-Zusammenfassung

Freie Universität Berlin

1. Eine Architektur beschreibt,
   - welche Teile ein System hat,
   - wie diese zusammenspielen
   - und wie dadurch die funktionalen und nichtfunktionalen Anforderungen erfüllt werden können

2. Zum Selbsterfinden von Architekturen gibt es wiederkehrende Architekturstile
   - Sollte man ungefähr kennen und verstehen

3. Insgesamt stützt man sich aber meist auf eine Standard-Architektur für den gegebenen Anwendungsbereich (Domäne)
   - Mindestens eine davon sollte man sehr genau kennen und verstehen
   - Das Selbsterfinden findet in der Regel nur lokal in einzelnen Komponenten statt

Literatur über Architektur

- Mary Shaw, David Garlan: *"Software Architecture: Perspectives on an Emerging Discipline"*, Prentice-Hall 1996
  - Das originale erste Buch zum Thema

- Johannes Siedersleben: *"Moderne Software-Architektur"*, dPunkt Verlag 2004
  - über QUASAR

- Es gibt natürlich viiiiiiel mehr

Nochmal: Wo sind wir gerade?

Freie Universität Berlin

Was müssen wir als nächstes leisten?

- Herausfinden **wie** man es bauen kann/sollte.
  D.h.:
- Entscheiden wie/wo/wodurch die funktionalen Anforderungen (Funktionen) umgesetzt werden
  - z.B. Technologieauswahl, Modulzerlegung, Wiederverwendung
  - Wir klammern Technologieauswahl und Wiederverwendung aus
  - Frage 1 ist also: **Wie zerlegt man ein System klug in Teile?**
- Herausfinden, wie man dabei die nichtfunktionalen Anforderungen alle erfüllen kann
  - viele nichtfunktionale Anforderungen haben globalen Charakter, werden also nicht von nur wenigen Modulen realisiert
  - Frage 2 ist also: **Wie findet man eine Gesamtstruktur mit den gewünschten globalen Eigenschaften?**

Systemzerlegung

Freie Universität Berlin

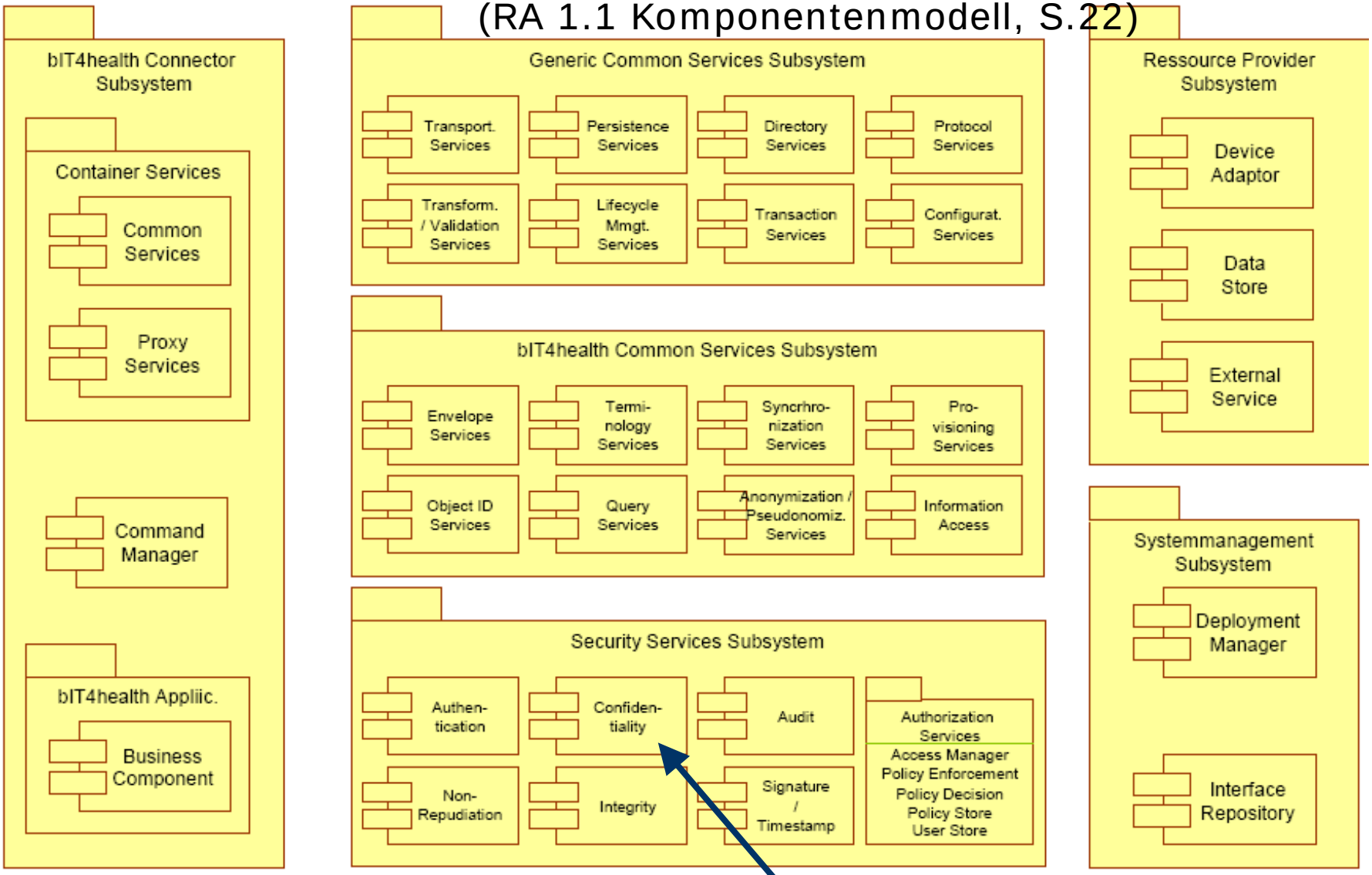
- Ist eine Architektur ausgewählt, muss das System hierarchisch in passende Teile zerlegt werden
  - Diese Teile heißen **Module** (neuerdings auch **Komponenten**)
  - Auf den oberen Ebenen (also für die großen Module) sagt man dazu auch oft **Subsysteme**

- Was gehört zusammen in ein Modul?
  - **Regel 1**: Die Inhaltsteile eines Moduls sollten viel mit einander zu tun haben (**Kohäsion**), aber statisch wenig mit den übrigen Modulen (**geringe Kopplung**)
    - hohe Kohäsion, sonst wären es keine vernünftigen "Teile"
    - geringe Kopplung, sonst ergäbe es keine richtige "Zerlegung"
  - **Regel 2**: Zusammen in ein Modul gehört all das, was aufgrund derselben Entwurfsentscheidung gestaltet wurde (und bei Korrektur dieser Entscheidung evtl. geändert werden muss).
    - Die Schnittstelle darf diese Entscheidung <u>nicht</u> sichtbar werden lassen (**Geheimnisprinzip**)

Beispiel: eGK-Subsysteme und -Komponenten

Freie Universität Berlin

(RA 1.1 Komponentenmodell, S.22)



Notation: UML Komponentendiagramm

z.B.: Entscheidung?

Begriff "Modul"

Freie Universität Berlin

- Ein Modul (in UML: Komponente) ist eine Einheit zusammengehörender Programmelemente
  - z.B. Daten, Datentypen, Klassen, Prozeduren, Funktionen, Prozesse etc.
  - Modulgrenzen sind oft (aber nicht immer) auch Grenzen programmiersprachlicher Konstrukte (wie Klasse, Paket etc.)
    - Eine Konstruktgrenze allein macht aber noch lange kein Modul

- Die Elemente gehören deshalb zusammen, weil sie von den selben Entwurfsentscheidungen betroffen sind
  - und das Modul verbirgt diese Entscheidungen vor dem Rest des Systems
  - Man nennt diese Entscheidungen die "**Geheimnisse**" des Moduls
  - Man nennt das Verbergen auch "**Verkapseln**" oder "**information hiding**"

Begriff "Modul" (2)

Freie Universität Berlin

- Modul ist nach außen nur üb. seine Schnittstelle beschrieben
  - Die Schnittstelle ist die Gesamtheit aller relevanten extern sichtbaren Eigenschaften des Moduls:
  - Die **Funktionen** (bei OO also auch **Klassen**) und ihre genaue garantierte **Semantik**
    - unbedingt also **Voraussetzungen** ("Vorbedingungen", preconditions) und **Effekte** (Wirkungen, "Nachbedingungen", postconditions),
      - in UML z.B. mittels OCL (object constraint language)
    - eventuell auch Eigenschaften wie Laufzeitverhalten, Speicherbedarf, Nebenläufigkeit, Kapazitätsgrenzen, Portabilität usw.

- Das Innere eines Moduls ist in der Regel komplizierter als die Schnittstelle
  - Dadurch reduziert Modularisierung die Systemkomplexität
    - Genau dies ist **der** Trick beim Entwurf
    - Kluge Schnittstellen zu definieren ist eine Kernkompetenz guter Softwareingenieure und -ingenieurinnen

Schnittstellen: Entwurf per Vertrag (Design by Contract)

Freie Universität Berlin

Spezifiziere für jede Operation

- **Voraussetzungen** ("Vorbedingungen", preconditions) und **Wirkungen** ("Nachbedingungen", postconditions)

- Diese Semantik definiert einen **Vertrag** zwischen Aufrufer und Modul:

  "Wenn Du, lieber Aufrufer,
  die Voraussetzungen einhältst,
  verspreche ich, die Wirkungen sicherzustellen"

  - Module so zu definieren wird *Entwurf per Vertrag (design by contract)* genannt
  - Die Verträge werden <u>vor</u> der Implementierung des Moduls festgelegt
    - sie stellen die Modulspezifikation dar

OCL-Beispiel: Methode Tournament.removePlayer

Freie Universität Berlin

context Tournament::removePlayer(p) **pre**:
isPlayerAccepted(p)

context Tournament::removePlayer(p) **post**:
not isPlayerAccepted(p)

context Tournament::removePlayer(p) **post**:
getNumPlayers() = @pre.getNumPlayers() – 1

- Wirkungen ("post") sind oft schwierig auszudrücken
- Aber auch unvollständige OCL-Beschreibungen können nützlich sein!
  - Ist obiges Beispiel vollständig?

Semantik von Schnittstellen und Implementierungen

Freie Universität Berlin

- Meist hat die Implementierung eines Moduls viele Eigenschaften, die nicht in der Schnittstelle beschrieben sind
- Auf solche Eigenschaften darf man sich außerhalb des Moduls <u>nicht</u> verlassen!
  - (denn sie gehören nicht zum Vertrag)
  - Sonst hat man einen inkorrekten Entwurf, weil die Eigenschaft nicht wie verlangt verborgen wird
- Beispiel folgt
- Leider werden Schnittst. nur selten so genau beschrieben, dass man dieses Prinzip wirklich einhalten kann

Beispiel: Semantik von Schnittstellen und Implementierungen

Freie Universität Berlin

Beispiel:

- Sie rufen nacheinander die Komponenten X und Y und wollen deren Fehlercode speichern, wenn eine davon versagt
  - und wollen später erkennen können, *welche* von beiden versagt hat
  - Schnittstelle von Y: definiert Fehlercodes 10 bis 20
  - Schnittstelle von X: definiert nur "non-zero indicates an error"
  - Aber Sie wissen, dass faktisch X dann stets nur 1 liefert

- Sie dürfen aber dieses Wissen nicht ausnutzen
  - und z.B. die Fehlercodes nicht in derselben Variablen ablegen
  - z.B. weil eine neue Version von X andere Codes benutzen könnte
  - und Sie dann X und Y evtl. nicht mehr unterscheiden könnten

(Falls Sie solches Wissen dennoch nutzen: Bitte verkapseln!)

Wirkung von Modulen/Modularisierung

Freie Universität Berlin

- Bei guten Entwürfen:
  - Die Schnittstellen verbergen gerade diejenigen Entwurfsentscheidungen, die sich öfter mal ändern
  - Bei späteren Änderungen muss dann meist nur ein Modul verändert werden, weil dessen Schnittstelle trotz der Änderung gleich bleiben kann
  - **Die Kunst beim Entwerfen liegt darin, solche Schnittstellen zu finden**

- Bei schlechten Entwürfen:
  - Es ist bei Änderungen oft auch die Schnittstelle betroffen und aufrufende Module müssen deshalb mit geändert werden
  - Das passiert vor allem dann, wenn man nicht vorhergesehen hat, was sich ändern könnte

Wirkung von Modulen/Modularisierung (2)

Freie Universität Berlin

- Bei guten Entwürfen:
  Das System ist einfacher zu verstehen
  - Weil man in jeder Situation von den meisten Modulen nur die Schnittstelle verstehen muss
  - aber nicht die dahinter verborgenen Geheimnisse und Details.
  - Modularisierung (d.h. Schnittstellenbildung) reduziert dadurch die effektive Komplexität des Ganzen.

- Bei schlechten Entwürfen:
  Es ist unklar, was alles zur Schnittstelle gehört
  - Deshalb fällt die Komplexitätsreduktion viel geringer aus,
  - weil man das Innere von Modulen betrachten muss,
    um die Schnittstelle zu ermitteln

Mythen über Module

Freie Universität Berlin

- "Ein Modul ist immer so circa 100-300 Zeilen lang"
  - Unsinn. Ein Modul kann sehr klein sein (10 Zeilen) oder sehr groß (1 Million Zeilen – dann enthält es aber sicherlich Teilmodule)
- "Eine Klasse ist ein Modul"
  - Nicht immer. Ein Modul kann gelegentlich kleiner sein als eine Klasse (z.B nur 1 oder 2 Methoden innerhalb einer Klasse) und ist in der Regel größer als eine Klasse (mehrere Klassen zusammen bilden das Modul)
- "Ein Java interface ist eine Modulschnittstelle"
  - Bei weitem nicht. Ein Java interface legt nur die Signaturen von Operationen eines Moduls fest.
    - Es fehlt eine Beschreibung der Bedeutung der Operationen
    - Es fehlt eine Beschreibung der nichtfunktionalen Eigenschaften
- "Module sind stets wiederverwendbar"
  - Nicht unbedingt. Module können sehr speziell für ein bestimmtes System sein. Sie kapseln lediglich eine Entwurfsentscheidung ein.

Modul: männlich oder sächlich?

Freie Universität Berlin

Das Modul (Plural: Die Module): [englisch: module]

- Technik:
  Baugruppe
  - etymologisch verwandt mit 'Modell' (und dessen Wurzeln sind sehr vielfältig)

Der Modul (Plural: Die Moduln): [englisch: modulus]

- Mathematik:
  Teilungsrest (Zahlen),
  eine Klasse algebraischer Strukturen
- Physik, Ingenieurwesen:
  Dimensionslose Maßzahl
  - etymologisch enger verwandt mit 'Maß' (gr. metron, lat. modus)

In der Softwaretechnik also immer: Das Modul, die Module

Wie findet man eine gute Modularisierung?

Freie Universität Berlin

- Es gibt nirgends eine wirkliche Anleitung, wie man das anstellt

- Sondern nur bekannte Kriterien zur Beurteilung des Resultats:
  - hohe Kohäsion innerhalb jedes Moduls
    - d.h. Module sind so klein wie sinnvoll möglich
  - geringe Kopplung zwischen den Modulen
  - Module sind meistens viel komplexer als ihre Schnittstellen
  - Im Übrigen sind die Entscheidungen verborgen, die sich am ehesten ändern

- Deshalb studieren wir das Problem in der nächsten Stunde an einem Fallbeispiel
  - KWIC – Keyword in Context

Zusammenfassung

Freie Universität Berlin

- Eine Architektur beschreibt,
  - welche **Teile** ein System hat,
  - wie diese **zusammenspielen**
  - und wie dadurch die **funktionalen** und **nichtfunktionalen Anforderungen** erfüllt werden können
- Wichtigster Aspekt einer Architektur ist die Modularisierung des Systems (Zerlegung in Module)
  - Ein Modul wird beschrieben über seine **Schnittstelle**
  - Die Schnittstelle verbirgt ein oder mehrere **Geheimnisse** des Moduls
  - Die Geheimnisse spiegeln **Entwurfsentscheidungen** wider, die sich ändern könnten
- Ein gut modularisiertes System ist einfach zu verstehen und Änderungen beschränken sich meist auf ein Modul

Danke!