

# Teil IV: Objektorientierter Entwurf (Object-Oriented Design, OOD)

Prof. Dr. rer. nat. Uwe Aßmann

Institut für Software- und  
Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

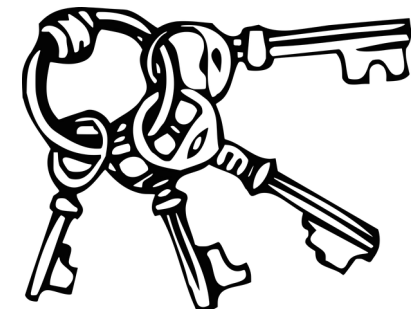
TU Dresden

Version 20-0.2, 27.06.20



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

# Das Ziel des Studiums (IV): Träume realisieren



- ▶ site for c/c++ counting tools <http://maultech.com/chrislott/resources/cmetrics/>

*No road is too long for him who advances slowly  
and does not hurry,*

*and no attainment is beyond his reach  
who equips himself with patience to achieve it.*

Jean de la Bruyère

[https://artistquoteoftheday.wordpress.com/2010/05/27/  
no-road-is-too-long-for-him-who-advances-  
slowly-and-does-not-hurry-and-no-attainment-is-  
beyond-his-reach-who-equips-himself-with-patience-to-achieve-it/](https://artistquoteoftheday.wordpress.com/2010/05/27/no-road-is-too-long-for-him-who-advances-slowly-and-does-not-hurry-and-no-attainment-is-beyond-his-reach-who-equips-himself-with-patience-to-achieve-it/)

# Teil IV - Objektorientierter Entwurf (Object-Oriented Design, OOD)

7

Softwaretechnologie (ST)

- 1) 40: Überblick
- 2) 41: Einführung in die objektorientierte Softwarearchitektur
  - 1) Architekturprinzipien, Architekturstile, Perspektivenmodelle
  - 2) Modularität und Geheimnisprinzip
  - 3) BCD-Architekturstil (3-tier architectures)
- 3) 42: Verfeinerung mit querschneidender Objektorichung
- 4) 43: Architektur interaktiver Systeme
- 5) 44: Punktweise Verfeinerung von Lebenszyklen
  - Verfeinerung von verschiedenen Steuerungsmaschinen

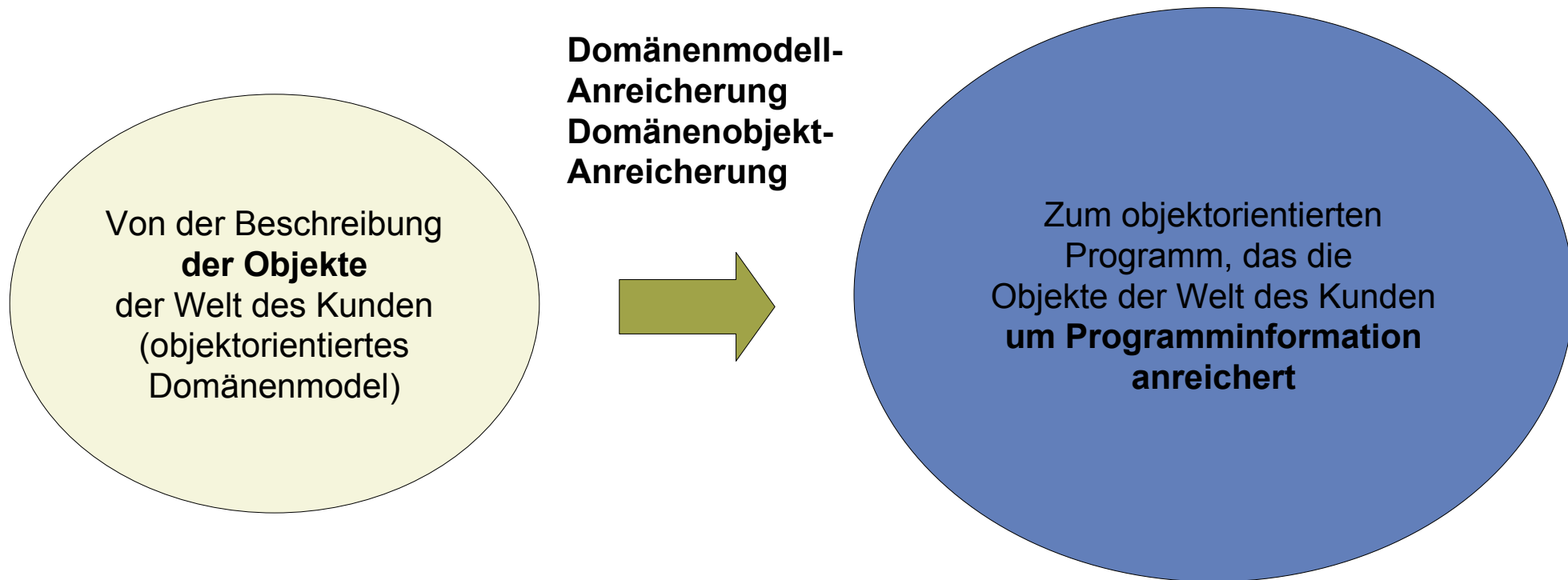


# 40.1. Verfeinerung des Anforderungsmodell zum Entwurfsmodell, Implementierungsmodell und Code



# Die zentralen Fragen des objektorientierten Ansatzes

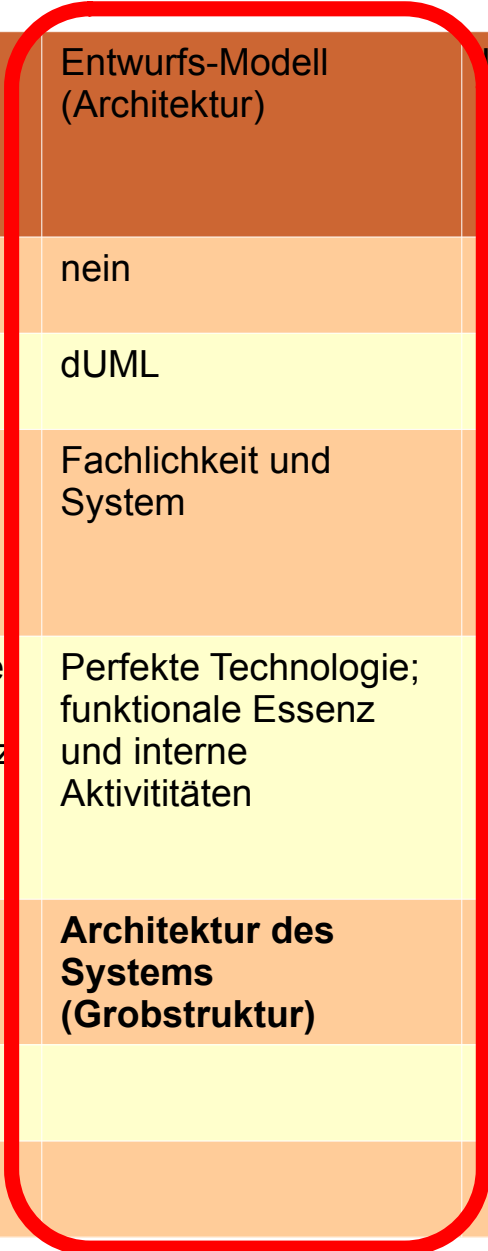
Wie kommen wir vom Problem des Kunden zum Programm (oder Produkt)?



Anreicherung/Verfettung: Anreicherung durch technische Programminformation  
„object fattening“: Anreicherung von Objekten des Domänenmodells durch Verfeinerungen

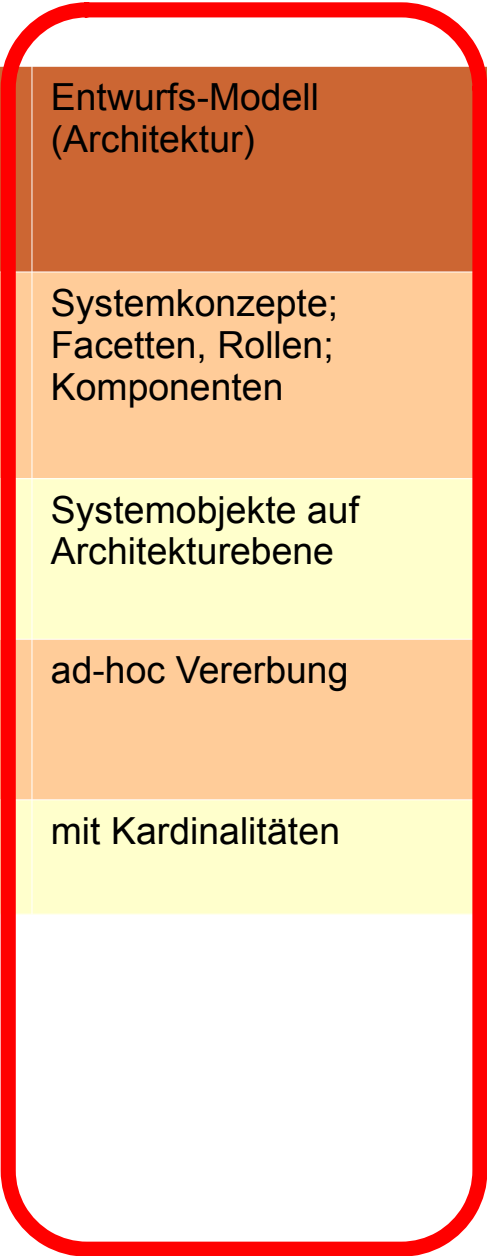
# Q5.1 Verfeinerung von UML von der Analyse zum Entwurf zur Implementierung

	Analyse-Modell	Entwurfs-Modell (Architektur)	Implementierungsmodell (Feinentwurf)	Implementierung (Programm)
vollständig	nein	nein	beinahe	ja
Sprache	aUML	dUML	UML	Java
Charakter	Fachlichkeit; Domäne	Fachlichkeit und System	Fachlichkeit, System und Technologie	Fachlichkeit, System, Technologie und Ressourcen
Technologie	Annahme perfekte Technologie; funktionale Essenz	Perfekte Technologie; funktionale Essenz und interne Aktivitäten	technologieabhängig (plattformabhängig)	technologieabhängig (plattformabhängig), ressourcenabhängig
Struktur	noch wenig	<b>Architektur des Systems (Grobstruktur)</b>	Präzise Spezifikation des Verhaltens des Systems	Lauffähiges Programm



# Q5.2 Verfeinerung der Sprachkonzepte

	Analyse-Modell	Entwurfs-Modell (Architektur)	Implementierungsmodell (Feinentwurf)	Implementierung
Klassen	Begriffe und Domänenkonzepte; Facetten, Rollen	Systemkonzepte; Facetten, Rollen; Komponenten	Systemkonzepte nur noch in einfachen Klassen; Rollen aufgelöst durch Entwurfsmuster	Systemkonzepte
Objekte	Domänenobjekte	Systemobjekte auf Architekturebene	Alle Systemobjekte	Alle Systemobjekte
Vererbung	Begriffshierarchien Mehrfachvererbung, Produktverbände	ad-hoc Vererbung	konform, 1-D-Vererbungshierarchie	konform
Assoziationen	oft ungerichtet	mit Kardinalitäten	abgeflacht	





# Verfeinerungsschritte beim Übergang zum Entwurfsmodell

- ▶ Siehe Kapitel 41
- ▶ Entwickeln des **Architekturaspektes** in der Schicht “Anwendungslogik”
  - Entwurfsmuster anwenden
  - Schichtung
  - Architekturprinzipien wie Geheimnisprinzip oder lose Kopplung
- ▶ Erhöhe **Wiederverwendbarkeit**
  - Ziehe ein Teamklassen, um Kollaborationen wiederverwendbar zu machen
  - Verwende Variabilitätsmuster
  - Verwende Erweiterungsmuster
  - Integriere externe Komponenten durch Klebemuster
  - Ausnutzen des Quasar-Prinzips zur Identifikation von wiederverwendbaren Einheiten
  - Einziehen von Komponenten mit angebotenen und benötigten Schnittstellen
  - Einziehen von Paketen zur Strukturierung
  - Einschränkung durch Sichtbarkeiten zur loserer Kopplung (Datenkapselung und Austauschbarkeit)

# Verfeinerungsschritte beim Übergang vom Analyse- zum Entwurfsmodell (II)

- ▶ **Vervollständigung fehlender Angaben (Kap. 42)**
  - Typisierung
  - Ausarbeitung von Objekt-Lebenszyklen (punktweise Verfeinerung)
  - Ausarbeitung von Kollaborationen (querschneidende Verfeinerung)
- ▶ **Detaillierung von zu groben Konzepten**
  - Querschneidende Objektorreicherung durch Kollaborationen
  - Assoziationen:
    - Einziehen von Navigationsrichtungen, um Platz zu sparen
    - Einziehen von Qualifikationen, um Suchen zu beschleunigen
    - Annotation von geordneten und sortierten Assoziationen
  - Einziehen von Schnittstellen zur Sicherstellung von homogenem Verhalten
  - Einziehen von Materialbehälterklassen (Verwaltungsklassen)

# Verfeinerungsschritte beim Übergang zum Entwurfsmodell (III)

- ▶ **Strukturierung** und Restrukturierung
  - Refactoring (Restrukturierung unter Beibehaltung der Semantik)
  - Identifikation von abgeleiteten Modellelementen zur Elimination von Redundanz (Verschlankung)
- ▶ **Flachklopfen** (Abflachen, Realisierung)
  - Integration von Unterobjekten (Rollen, Facetten, Teile) mit Hilfe von Entwurfsmustern: Compositum, Brücke, u.v.m.
  - Realisierung von Objektnetzen: Abflachen der Assoziationen
- ▶ **Erhöhung Zuverlässigkeit**
  - Verfeinerung der Vererbungsrelation zu konformer Vererbung zur Elimination von Fehlern
  - Schreiben von Verträgen mit Vor- und Nachbedingungen

# Schritte beim Übergang zum Implementierungsmodell (Feinentwurf)

- ▶ Vervollständigung fehlender Angaben
  - Implementierung von Methoden
- ▶ Strukturierung und Restrukturierung
  - Auflösen von Mehrfachvererbung in Aggregation
  - Einziehen von abgeleiteten Relationen, um Zugriffe, Navigationen und Abfragen zu beschleunigen (Verfettung)
- ▶ Detaillierung (Implementierungsmuster anwenden)
  - Assoziationen:
    - Einziehen von Navigationsrichtungen, um Platz zu sparen; Ersetzen durch Suchalgorithmen
    - Abbildung von qualifizierten Assoziationen, um Suchen zu beschleunigen
    - Annotation von geordneten und sortierten Assoziationen
  - Verfeinerung der Vererbungsrelation zu *konformer Vererbung* zur Elimination von Fehlern
  - Transformation in die Implementierungssprache
- Verhalten angeben
  - Lebenszyklen modellieren
  - Implementierung von Schnittstellen auswählen

# Schritte beim Übergang zum Code

- ▶ **Codegenerierung:** Nutzung von Codegenerator-Werkzeugen, um
  - Architekturcode zu generieren
  - Objekt-Lebenszyklen in Programm zu übersetzen
  - Nutzung von Webe-Werkzeugen, um querschneidende Kollaborationen in Klassen einzuweben
- ▶ Alternativ: Umsetzung der Modelle in Code von Hand
- ▶ Hand-Implementierung fehlender Komponenten

## 40.2. Herstellung Großer Softwaresysteme

*software*: computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system.

(IEEE Standard Glossary of Software Engineering)

# Was heißt hier "groß"?

- ▶ Klassifikation nach W. Hesse:

Klasse	Anzahl Code-Zeilen	Personenjahre zur Entwicklung
sehr klein	bis 1.000	bis 0,2
Klein	1.000 - 10.000	0,2 - 2
mittel	10.000 - 100.000	2 - 20
groß	100.000 - 1 Mio.	20 - 200
sehr groß	über 1 Mio.	über 200

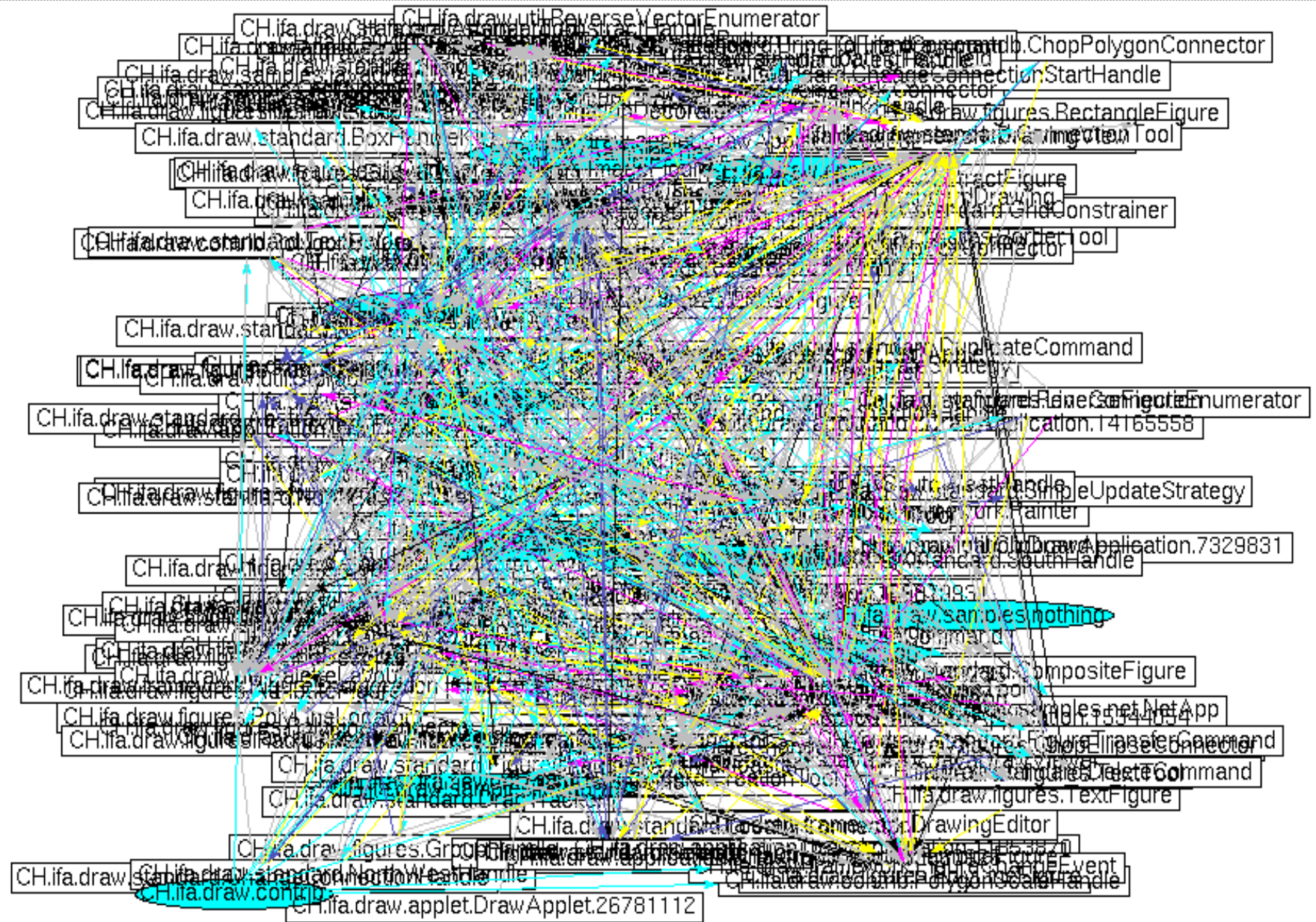
# Riesige Systeme

- ▶ Telefonvermittlungssoftware EWSD Siemens (Version 8.1): <https://de.wikipedia.org/wiki/EWSD>
  - 12,5 Mio. Code-Zeilen
  - ca. 6000 Personenjahre
- ▶ Umfang der verwendeten Software (Anfang 2000):
  - SAP Enterprise Resource Planning (ERP)-Software R/3 (Version 4.0): 50 Mio. Code-Zeilen
  - Credit Suisse 25 Mio. Code-Zeilen
  - Citicorp Bank: 400 Mio. Code-Zeilen
  - AT&T: 500 Mio. Code-Zeilen
  - General Motors: 2 Mrd. Code-Zeilen
  - Microsoft, Windows Server 50 Mio (2003)
- ▶ HEUTE:
  - Mercedes Benz S-Klasse 9 Mio LOC
  - Credit Suisse > 100 Mio LOC
  - Google 2 Mrd. LOC
  - <https://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/>



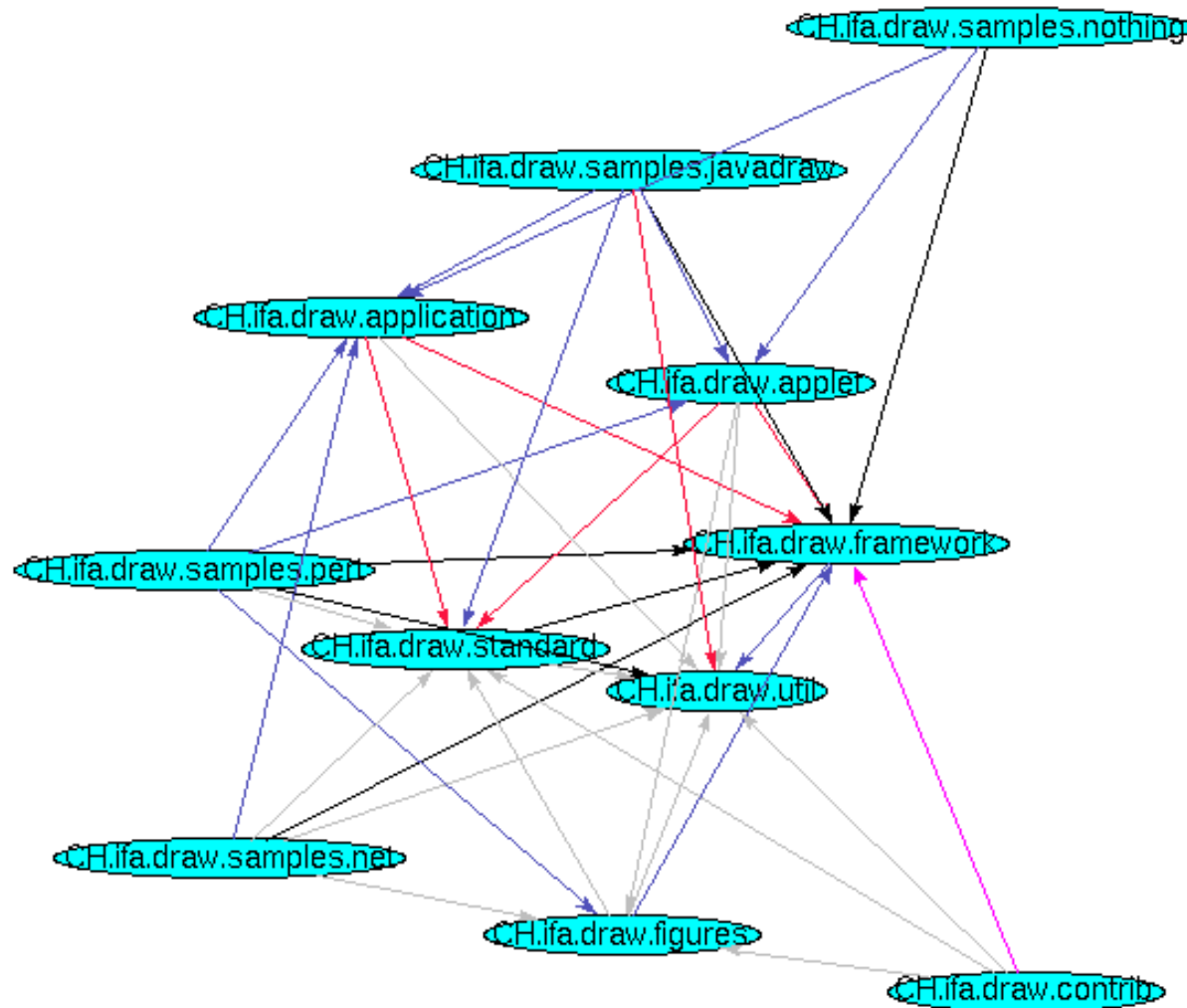
# Strukturprobleme

- ▶ Folgende Bilder stammen vom Goose Reengineering-Werkzeug aus der Analyse eines Java-Systems [Goose, FZI Karlsruhe, <http://www.fzi.de>]





# Gefaltet





# Die Bedeutung von Softwarearchitektur

- ▶ Softwarearchitektur ist der Schlüssel zum Erfolg des Softwareingenieurs und seiner Firma.

**Ohne** gute Softwarearchitektur keine:  
Zuverlässigkeit, Einbruchssicherheit,  
Wiederverwendung, Variabilität,  
Evolution, Erweiterbarkeit

**Mit** guter Softwarearchitektur:  
Softwareproduktlinien, schnell erstellte neue Produkte,  
vertikale Portierung auf andere Domänen,  
einfaches Dienstleistungsgeschäft.

# The End

## Teil IV: Objektorientierter Entwurf (Object-Oriented Design, OOD)

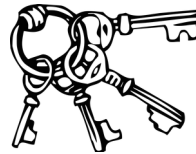
Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 20-0.2, 27.06.20



## • Parallelen zum Fachgebiet der Architektur:

- Architekten sind an der Nahtstelle zwischen Kunde und Baufirma.
- Schlechter Architekturentwurf kann nicht durch gute Bauqualität kompensiert werden.
- Es gibt Architektur-Spezialisten für bestimmte Anwendungsgebiete.
- Es gibt "Schulen", die bestimmte Grundprinzipien vertreten.

## Das Ziel des Studiums (IV): Träume realisieren





- ▶ site for c/c++ counting tools <http://maultech.com/chrislott/resources/cmmetrics/>

*No road is too long for him who advances slowly  
and does not hurry,*

*and no attainment is beyond his reach  
who equips himself with patience to achieve it.*

Jean de la Bruyère

[https://artistquoteoftheday.wordpress.com/2010/05/27/  
no-road-is-too-long-for-him-who-advances-  
slowly-and-does-not-hurry-and-no-attainment-is-  
beyond-his-reach-who-equips-himself-with-patience-to-achieve-it/](https://artistquoteoftheday.wordpress.com/2010/05/27/no-road-is-too-long-for-him-who-advances-slowly-and-does-not-hurry-and-no-attainment-is-beyond-his-reach-who-equips-himself-with-patience-to-achieve-it/)

## Teil IV - Objektorientierter Entwurf (Object-Oriented Design, OOD)

7 Softwaretechnologie (ST)

- 1) 40: Überblick
- 2) 41: Einführung in die objektorientierte Softwarearchitektur
  - 1) Architekturprinzipien, Architekturstile, Perspektivenmodelle
  - 2) Modularität und Geheimnisprinzip
  - 3) BCD-Architekturstil (3-tier architectures)
- 3) 42: Verfeinerung mit querschneidender Objektorichung
- 4) 43: Architektur interaktiver Systeme
- 5) 44: Punktweise Verfeinerung von Lebenszyklen
  - Verfeinerung von verschiedenen Steuerungsmaschinen



## 40.1. Verfeinerung des Anforderungsmodell zum Entwurfsmodell, Implementierungsmodell und Code



**Softwaresystem:** Ein System (oder Teilsystem), dessen Komponenten aus Software bestehen

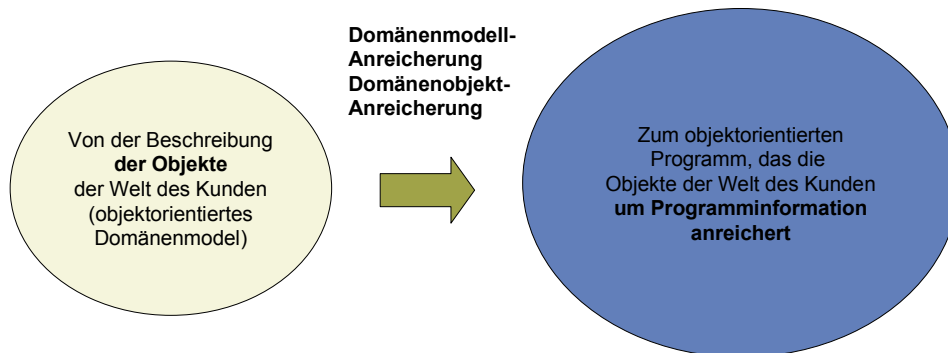
Klassifikation von Software:

- Generisches Produkt oder Einzelanfertigung (vereinbartes Produkt)?
- Systemsoftware oder Anwendungssoftware?
- eingebettete (produktintegriert) oder eigenständig?
- Realzeitanforderungen oder flexiblere Zeitanforderungen?
- Datenintensiv oder berechnungsintensiv?
- Einzel- oder Mehrfachbenutzung?

Software ist ein immaterielles Produkt.

# Die zentralen Fragen des objektorientierten Ansatzes

Wie kommen wir vom Problem des Kunden zum Programm (oder Produkt)?



## Q5.1 Verfeinerung von UML von der Analyse zum Entwurf zur Implementierung

	Analyse-Modell	Entwurfs-Modell (Architektur)	Implementierungsmodell (Feinentwurf)	Implementierung (Programm)
vollständig	nein	nein	beinahe	ja
Sprache	aUML	dUML	UML	Java
Charakter	Fachlichkeit; Domäne	Fachlichkeit und System	Fachlichkeit, System und Technologie	Fachlichkeit, System, Technologie und Ressourcen
Technologie	Annahme perfekte Technologie; funktionale Essenz	Perfekte Technologie; funktionale Essenz und interne Aktivitäten	technologieabhängig (plattformabhängig)	technologieabhängig (plattformabhängig), ressourcenabhängig
Struktur	noch wenig	<b>Architektur des Systems (Grobstruktur)</b>	Präzise Spezifikation des Verhaltens des Systems	Lauffähiges Programm

## Q5.2 Verfeinerung der Sprachkonzepte

	Analyse-Modell	Entwurfs-Modell (Architektur)	Implementierungsmodell (Feinentwurf)	Implementierung
Klassen	Begriffe und Domänenkonzepte; Facetten, Rollen	Systemkonzepte; Facetten, Rollen; Komponenten	Systemkonzepte nur noch in einfachen Klassen; Rollen aufgelöst durch Entwurfsmuster	Systemkonzepte
Objekte	Domänenobjekte	Systemobjekte auf Architekturebene	Alle Systemobjekte	Alle Systemobjekte
Vererbung	Begriffshierarchien Mehrfachvererbung, Produktverbände	ad-hoc Vererbung	konform, 1-D-Vererbungshierarchie	konform
Assoziationen	oft ungerichtet	mit Kardinalitäten	abgeflacht	



# Verfeinerungsschritte beim Übergang zum Entwurfsmodell

- ▶ Siehe Kapitel 41
- ▶ Entwickeln des **Architekturaspektes** in der Schicht “Anwendungslogik”
  - Entwurfsmuster anwenden
  - Schichtung
  - Architekturprinzipien wie Geheimnisprinzip oder lose Kopplung
- ▶ Erhöhe **Wiederverwendbarkeit**
  - Ziehe ein Teamklassen, um Kollaborationen wiederverwendbar zu machen
  - Verwende Variabilitätsmuster
  - Verwende Erweiterungsmuster
  - Integriere externe Komponenten durch Klebemuster
  - Ausnutzen des Quasar-Prinzips zur Identifikation von wiederverwendbaren Einheiten
  - Einziehen von Komponenten mit angebotenen und benötigten Schnittstellen
  - Einziehen von Paketen zur Strukturierung
  - Einschränkung durch Sichtbarkeiten zur loseren Kopplung (Datenkapselung und Austauschbarkeit)



## Verfeinerungsschritte beim Übergang vom Analyse- zum Entwurfsmodell (II)

- ▶ **Vervollständigung fehlender Angaben (Kap. 42)**
  - Typisierung
  - Ausarbeitung von Objekt-Lebenszyklen (punktweise Verfeinerung)
  - Ausarbeitung von Kollaborationen (querschnittende Verfeinerung)
- ▶ **Detaillierung von zu groben Konzepten**
  - Querschnittende Objektanreicherung durch Kollaborationen
  - Assoziationen:
    - Einziehen von Navigationsrichtungen, um Platz zu sparen
    - Einziehen von Qualifikationen, um Suchen zu beschleunigen
    - Annotation von geordneten und sortierten Assoziationen
  - Einziehen von Schnittstellen zur Sicherstellung von homogenem Verhalten
  - Einziehen von Materialbehälterklassen (Verwaltungsklassen)





## Verfeinerungsschritte beim Übergang zum Entwurfsmodell (III)

- ▶ **Strukturierung** und Restrukturierung
  - Refactoring (Restrukturierung unter Beibehaltung der Semantik)
  - Identifikation von abgeleiteten Modellelementen zur Elimination von Redundanz (Verschlankung)
- ▶ **Flachklopfen** (Abflachen, Realisierung)
  - Integration von Unterobjekten (Rollen, Facetten, Teile) mit Hilfe von Entwurfsmustern: Compositum, Brücke, u.v.m.
  - Realisierung von Objektnetzen: Abflachen der Assoziationen
- ▶ **Erhöhung Zuverlässigkeit**
  - Verfeinerung der Vererbungsrelation zu konformer Vererbung zur Elimination von Fehlern
  - Schreiben von Verträgen mit Vor- und Nachbedingungen



# Schritte beim Übergang zum Implementierungsmodell (Feinentwurf)

- ▶ Vervollständigung fehlender Angaben
  - Implementierung von Methoden
- ▶ Strukturierung und Restrukturierung
  - Auflösen von Mehrfachvererbung in Aggregation
  - Einziehen von abgeleiteten Relationen, um Zugriffe, Navigationen und Abfragen zu beschleunigen (Verfettung)
- ▶ Detaillierung (Implementierungsmuster anwenden)
  - Assoziationen:
    - Einziehen von Navigationsrichtungen, um Platz zu sparen; Ersetzen durch Suchalgorithmen
    - Abbildung von qualifizierten Assoziationen, um Suchen zu beschleunigen
    - Annotation von geordneten und sortierten Assoziationen
  - Verfeinerung der Vererbungsrelation zu *konformer Vererbung* zur Elimination von Fehlern
  - Transformation in die Implementierungssprache
- Verhalten angeben
  - Lebenszyklen modellieren
  - Implementierung von Schnittstellen auswählen



## Schritte beim Übergang zum Code

- ▶ **Codegenerierung:** Nutzung von Codegenerator-Werkzeugen, um
  - Architekturcode zu generieren
  - Objekt-Lebenszyklen in Programm zu übersetzen
  - Nutzung von Web-Werkzeugen, um querschnittende Kollaborationen in Klassen einzuweben
- ▶ Alternativ: Umsetzung der Modelle in Code von Hand
- ▶ Hand-Implementierung fehlender Komponenten



## 40.2. Herstellung Großer Softwaresysteme

*software*: computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system.

(IEEE Standard Glossary of Software Engineering)

**Softwaresystem:** Ein System (oder Teilsystem), dessen Komponenten aus Software bestehen

Klassifikation von Software:

- Generisches Produkt oder Einzelanfertigung (vereinbartes Produkt)?
- Systemsoftware oder Anwendungssoftware?
- eingebettete (produktintegriert) oder eigenständig?
- Realzeitanforderungen oder flexiblere Zeitanforderungen?
- Datenintensiv oder berechnungsintensiv?
- Einzel- oder Mehrfachbenutzung?

Software ist ein immaterielles Produkt.

## Was heißt hier "groß"?

- ▶ Klassifikation nach W. Hesse:

Klasse	Anzahl Code-Zeilen	Personenjahre zur Entwicklung
sehr klein	bis 1.000	bis 0,2
Klein	1.000 - 10.000	0,2 - 2
mittel	10.000 - 100.000	2 - 20
groß	100.000 - 1 Mio.	20 - 200
sehr groß	über 1 Mio.	über 200

Quelle: Gumm/Sommer, Einführung in die Informatik, 4. Auflage, 2000, S. 639

Annahme: eine Person produziert pro Tag ca. 10 bis 100 Zeilen Programmcode, d.h. ca. 5000 Zeilen im Jahr.

## Riesige Systeme

- ▶ Telefonvermittlungssoftware EWSD Siemens (Version 8.1): <https://de.wikipedia.org/wiki/EWSD>
  - 12,5 Mio. Code-Zeilen
  - ca. 6000 Personenjahre
- ▶ Umfang der verwendeten Software (Anfang 2000):
  - SAP Enterprise Resource Planning (ERP)-Software R/3 (Version 4.0): 50 Mio. Code-Zeilen
  - Credit Suisse 25 Mio. Code-Zeilen
  - Citicorp Bank: 400 Mio. Code-Zeilen
  - AT&T: 500 Mio. Code-Zeilen
  - General Motors: 2 Mrd. Code-Zeilen
  - Microsoft, Windows Server 50 Mio (2003)
- ▶ HEUTE:
  - Mercedes Benz S-Klasse 9 Mio LOC
  - Credit Suisse > 100 Mio LOC
  - Google 2 Mrd. LOC
    - <https://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/>

Quellen für die Zahlen:

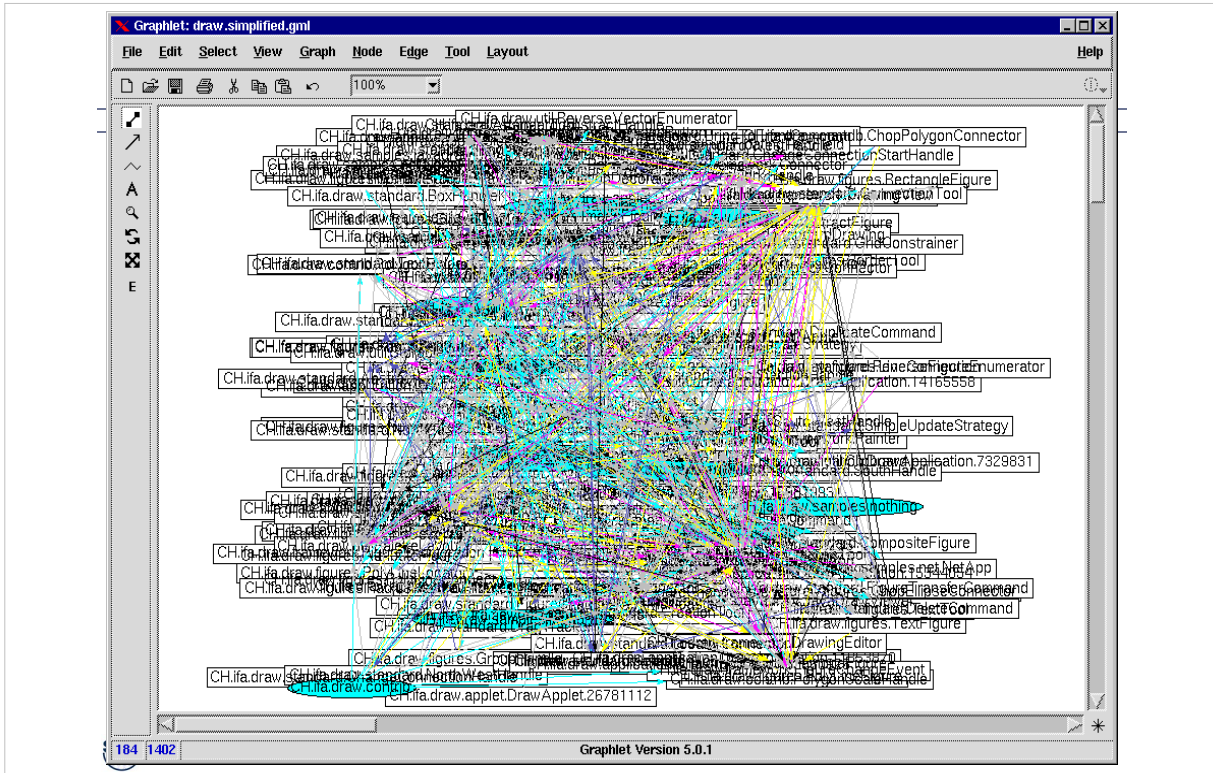
EWSD: Siemens-interne Information

SAP: [www.ct-software.com](http://www.ct-software.com)

Jahr 2000: Gary North's Y2K Links and Forums, basierend auf Presseberichten zwischen 1996 und 1999

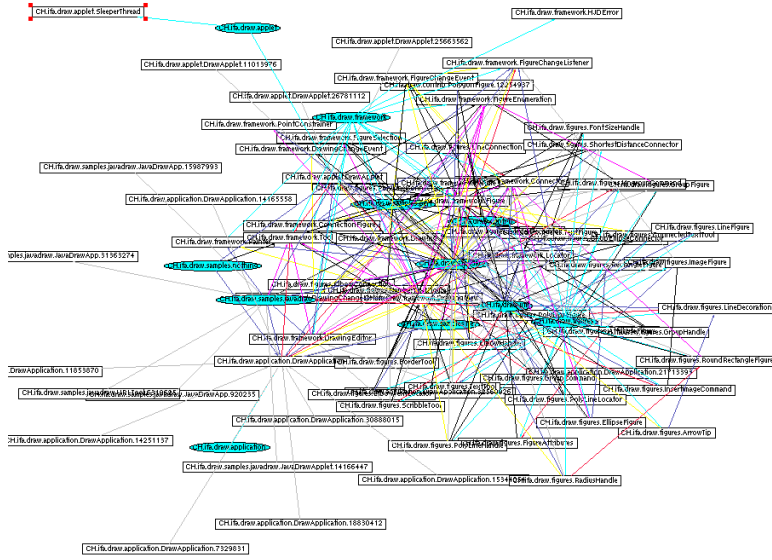
Quellcode von Windows 2000 (gerüchteweise): ca. 50-60 Mio. LOC

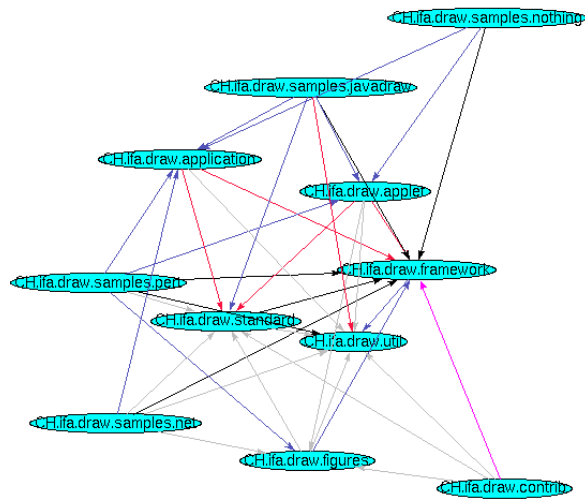
- ▶ Folgende Bilder stammen vom Goose Reengineering-Werkzeug aus der Analyse eines Java-Systems [Goose, FZI Karlsruhe, <http://www.fzi.de>]





# Teilweise gefaltet





# Die Bedeutung von Softwarearchitektur

- ▶ Softwarearchitektur ist der Schlüssel zum Erfolg des Softwareingenieurs und seiner Firma.

**Ohne gute Softwarearchitektur keine:**

Zuverlässigkeit, Einbruchssicherheit,  
Wiederverwendung, Variabilität,  
Evolution, Erweiterbarkeit

**Mit guter Softwarearchitektur:**

Softwareproduktlinien, schnell erstellte neue Produkte,  
vertikale Portierung auf andere Domänen,  
einfaches Dienstleistungsgeschäft.



# The End