Entwicklung und Analyse von Software-Architekturen

illustriert am Beispiel von Übersetzern

Dr. Peter Knauber

Fraunhofer Institut für
Experimentelles Software Engineering IESE

22. Juni 1998

Inhaltsübersicht

- Architektur von Software-Systemen
- Analyse von Software-Architekturen
- Entwicklung der Architektur von Übersetzern
- Zusammenfassung und Ausblick

Architektur von Software-Systemen

Was ist eine „Software-Architektur”?

- ein Überblick über die Struktur eines Systems

- die Repräsentation der grundlegenden
  - Konzepte und
  - deren Beziehungen

  aus der „realen Welt” in der Modellwelt;
  sie beschreibt
  - Komponenten (Components),
  - Verbindungen (Connections) und deren
  - Konfiguration (Configuration → „C^3“)

Beschreibung von Software-Architekturen

Informell:

- „Kästchen und Pfeile” (intuitiv)
- Text
- Kombinationen aus beiden

Formal:

Architektur-Beschreibungssprachen, z.B.:

- Rapide [Luckham et al.93]
- UniCon, Aesop, MetaH, Darwin, C2, …

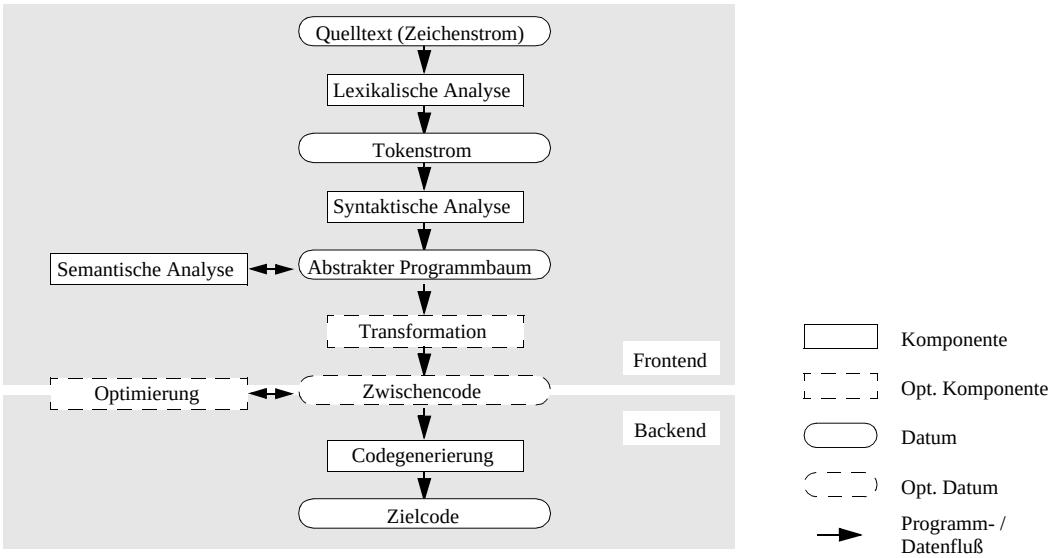
Hybrid:

z.B. „4+1 View Model” [Kruchten95]:

- Logical View: Funktionale Anforderungen (z.B. C^3)
- Process View: Parallelität, Synchronisation, …
- Physical View: Abbildung von Software auf Hardware
- Development View: Organisation von Software-Modulen
- Scenarios: Illustration, Integration der anderen Sichten

Vorteile / Nachteile

Beispiel für eine Software-Architektur: Übersetzer



→ Interpretierbar!

Analyse von Software-Architekturen

Nutzen von Software-Architekturen

- Erklären der Arbeitsweise eines Systems
- Analyse der Qualität eines Software-Systems:
  → in Bezug auf gegenwärtige Anforderungen
  → in Bezug auf potentielle, d.h. vermutete, zukünftige Anforderungen

Fragen:

- Wie beschreibt man Qualitätsanforderungen, um ein Software-System auf Architekturebene zu beurteilen?
- Wie beschreibt man Qualitätsanforderungen?

Beispiele für Qualitätsanforderungen

funktional:

- Aufgaben, die zu lösen sind
- Geschwindigkeit
- Bedienbarkeit
- Benutzungsoberfläche, Betriebssystem

→ „Anforderungen" (→ Pflichtenheft usw.)

nicht-funktional:

- erweiterbar
- modifizierbar
- wartbar
- portierbar
- skalierbar

→ ???

Nicht-funktionale Qualitätsanforderungen

Diese sind zu abstrakt, Beispiel:

„Bedienbarkeit" eines Window-Managers:

- die Schriftart, z.B. für Fenstertitel,
  kann durch ein Menü eingestellt werden

- die Hintergrundfarbe
  kann nur durch Neu-Übersetzen beeinflußt werden

→ Qualitätsanforderungen an ein System können nicht abstrakt beschrieben werden, sondern nur im Kontext
- seiner Benutzer und
- der Organisation (Arbeitsumfeld).

Szenarien zum Beschreiben von Qualitätsanforderungen

- Szenarien (use cases) beschreiben konkrete Anforderungen in einem bestimmten Kontext
  Beispiele:
  - Es muß möglich sein, im Window-Manager zur Laufzeit die Schrift zu verändern.
  - Der Compiler soll die Quellsprache X in die Zielsprache Y übersetzen.
    Er soll auf der Maschine / dem Betriebssystem Z laufen.

Fragen:

- Gliederung, Struktur, Granularität der Szenarien
- Gewichtung, Vergeben von Prioritäten
- Aufspüren, Erfassen von Szenarien

Aufspüren von Szenarien

Lösungsansatz:

Es werden Repräsentanten aller Rollen befragt, die an der Entwicklung oder dem Einsatz eines Systems beteiligt sind.

Rollen im Übersetzerbau:

- Sprachtheoretiker, Sprachentwickler
- Entwickler des Übersetzers
- Wartungspersonal
- Administrator der Einsatzumgebung des Übersetzers
- Benutzer des Übersetzers
- Benutzer des erzeugten Zielcodes

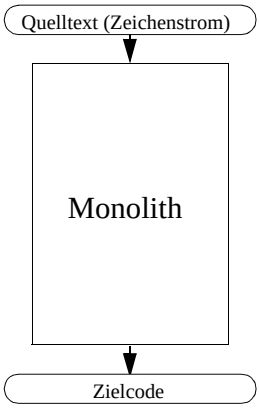
Szenarien für den Übersetzerbau

Benutzer des Übersetzers	Übersetzen eines Programms aus Sprache X in Sprache Y
Sprachentwickler	Lexikalische und syntaktische Analyse mit einer geeigneten Sprache beschreiben
Entwickler des Übersetzers, Wartungspersonal	Komplexität des Übersetzers reduzieren
Wartungspersonal, (beide) Benutzer	Analyseteil auch für andere Zielrechner benutzen
Benutzer des Zielcodes	Zielcode soll schneller laufen
Benutzer des Übersetzers	Integrierte Benutzung unterstützender Werkzeuge
Sprachentwickler	Übersetzer schnell zur Verfügung haben

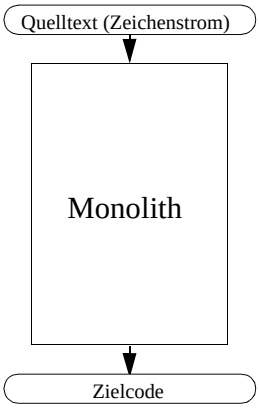
Entwicklung der Architektur von Übersetzern

Erstes Szenario: Benutzer des Übersetzers: Übersetzen eines Programms aus Sprache X in Sprache Y

Ergebnis: Monolithisches System



Ausgangssituation:



Sprachentwickler:

Lexikalische und syntaktische Analyse mit einer geeigneten Sprache beschreiben

Entwickler des Übersetzers, Wartungspersonal:

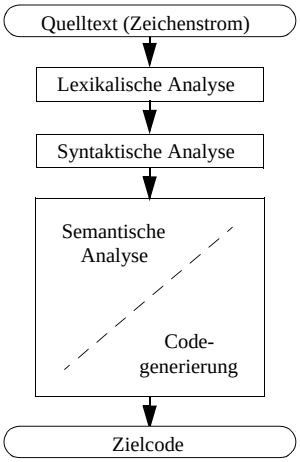
Komplexität des Übersetzers reduzieren

Konsequenz:

→ Reguläre Ausdrücke zur Beschreibung der lexikalischen Analyse, getrennte Scanner-Komponente

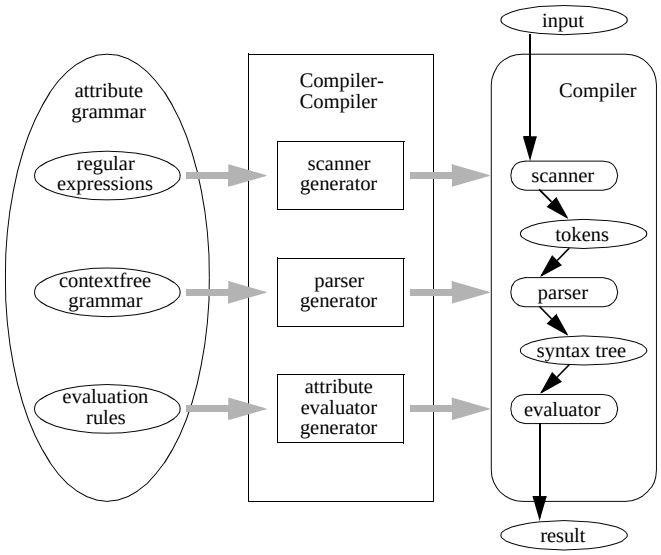
→ Kontextfreie Grammatik zur Beschreibung der syntaktischen Analyse, getrennte Parser-Komponente

Entwicklung von Scanner und Parser (-Generatoren)

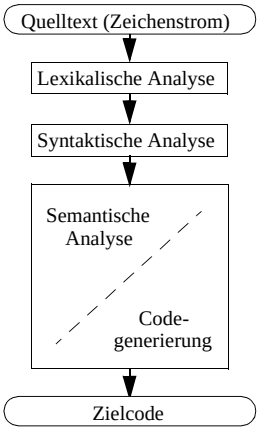


Smalltalk-basierter Interaktiver Compiler-Compiler [Schmitz et al.]

SIC



Ausgangssituation:



Wartungspersonal, (beide) Benutzer:

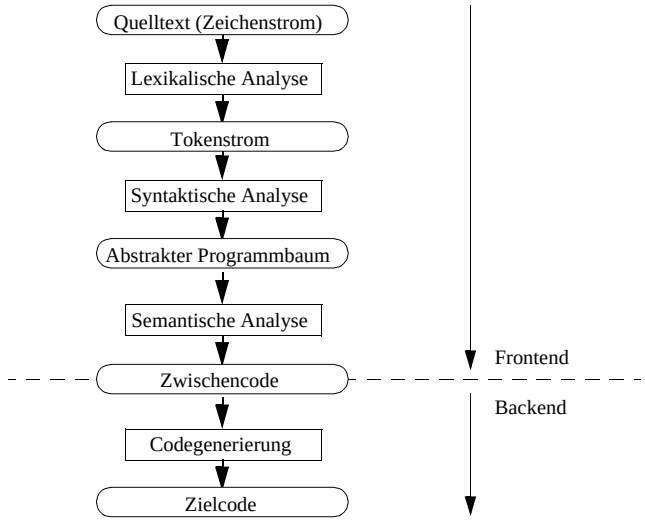
Analyseteil auch für andere Zielrechner benutzen

Konsequenz:

→ Semantische Analyse von der Codegenerierung trennen

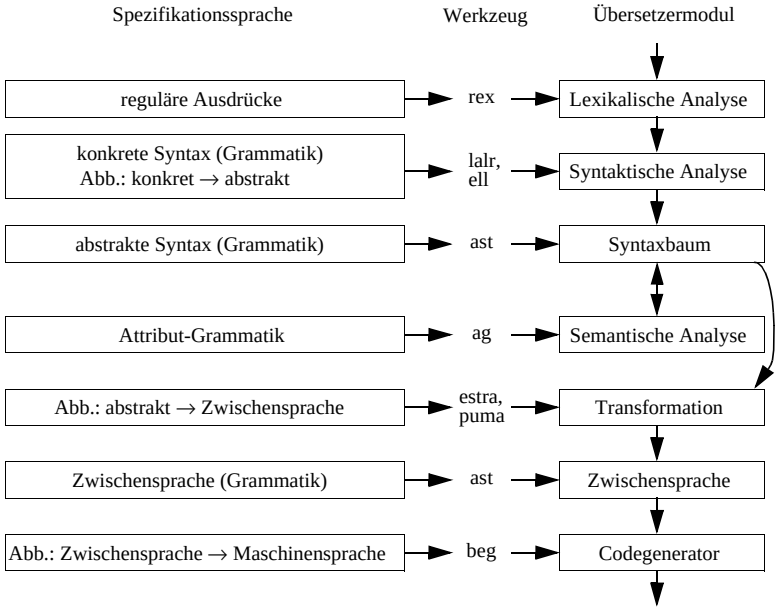
Splitten des Übersetzers in „Frontend” und „Backend”

Separates Backend



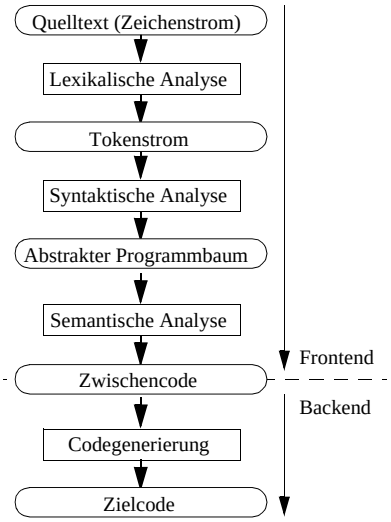
Fraunhofer IESE
Einrichtung
Experimentelles
Software Engineering

Compiler-Toolbox Cocktail [GroschEmmelmann90]



Fraunhofer IESE
Einrichtung
Experimentelles
Software Engineering

Ausgangssituation:



Benutzer des Zielcodes:

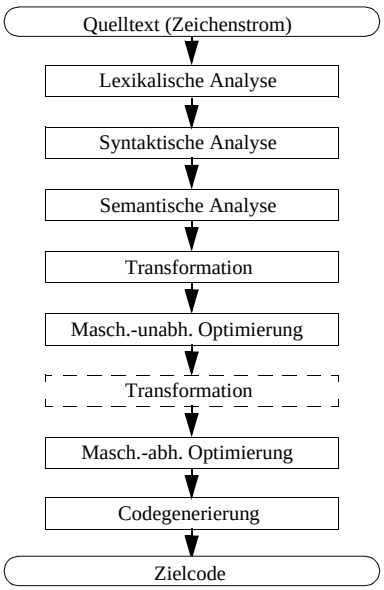
Zielcode soll schneller laufen

Konsequenz:

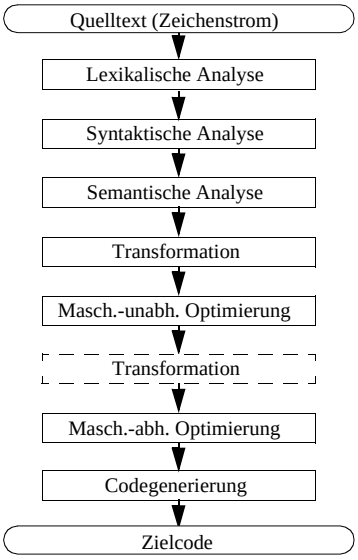
→ Optimieren zwischen semantischer Analyse und Codegenerierung:

- maschinenunabhängig
- maschinenabhängig

Optimierungsphase(n)



Ausgangssituation:



Benutzer des Übersetzers:

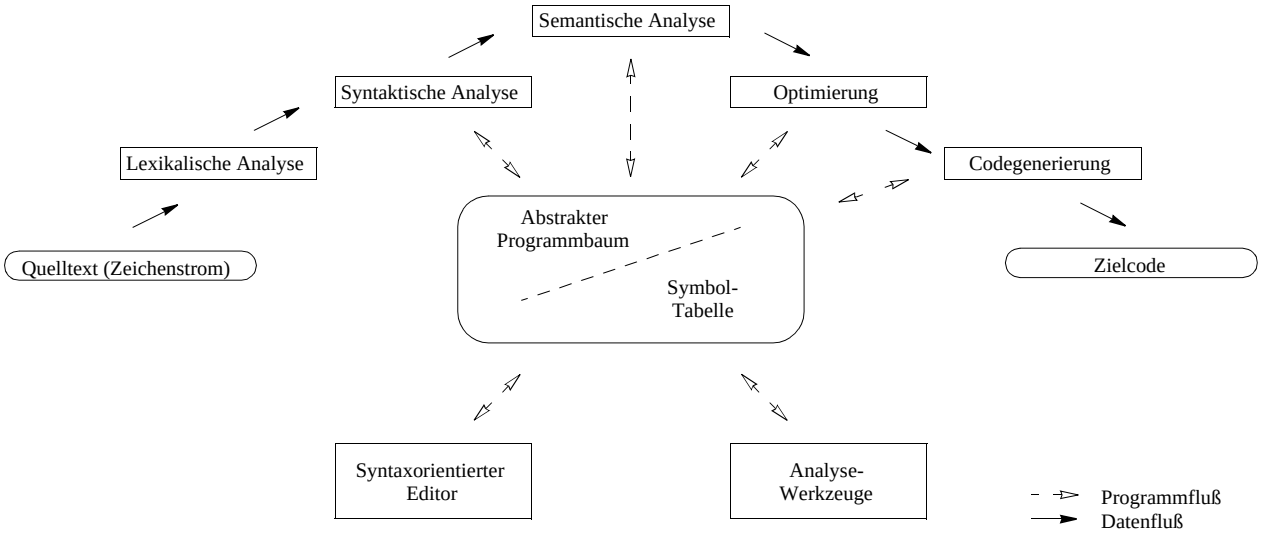
Integrierte Benutzung unterstützender Werkzeuge

Konsequenz:

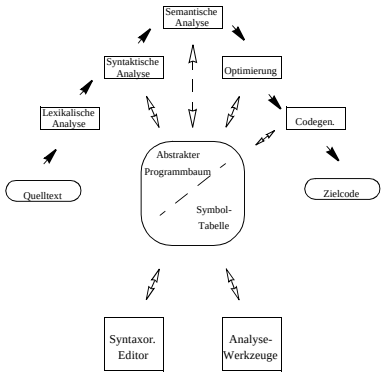
→ Trennen von

- Daten und Routinen zu deren Bearbeitung (besonders Namensverwaltung) und
- Programm

Integrierte Entwicklungsumgebung (datenzentriert)



Ausgangssituation:



Sprachentwickler:

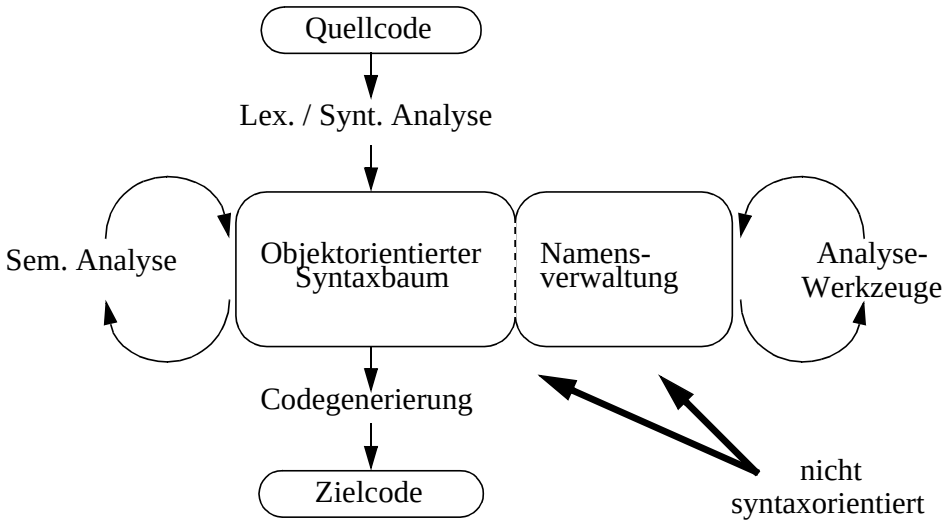
Übersetzer schnell zur Verfügung haben
(um neue Sprachen kontrolliert entwickeln und testen zu können)

Konsequenz:

Semantik und Codegenerierung sind am aufwendigsten zu implementieren

→ als Bausteine wiederverwenden

Modularer Aufbau der sem. Analyse und der Codegenerierung



Zusammenfassung und Ausblick

Bemerkungen

- Scanner und Parser, d.h. ihre Beschreibung und Aufgaben, haben sich nicht mehr (wesentlich) verändert (→ basieren auf theoretischen Erkenntnissen)
- Es gibt keine gemeinsame Struktur für die Namensverwaltung / Symboltabelle, aber:
  Die innere Struktur der Übersetzerkomponenten gleicht sich immer mehr

Bisher: Evolutionäre Entwicklung

Idee: Zukünftige Entwicklung von Systemen abschätzen / planen

→ Architektur ist die geeignete Abstraktionsebene für derartige Überlegungen

Ausblick

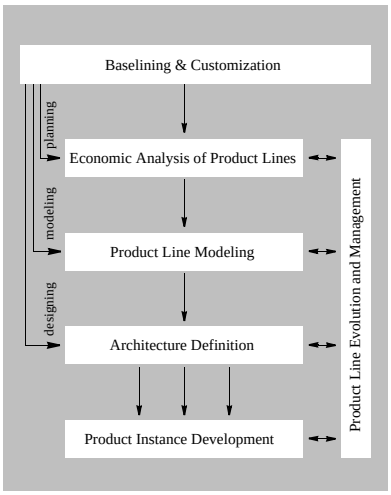
Ziel: Produktlinien für Software-Systeme etablieren
(wie sie für andere Branchen bereits lange und erfolgreich existieren)
indem
ganze Anwendungsbereiche (Domänen) statt Einzelsystemen
betrachtet und beschrieben werden

Grundlagen:
- Architektur-Beobachtung und -Analyse
- Entwicklung domänenspezifischer Software-Architekturen (DSSA) im Rahmen von „Software Technology for Adaptable and Reliable System" (STARS, [Trimble94])

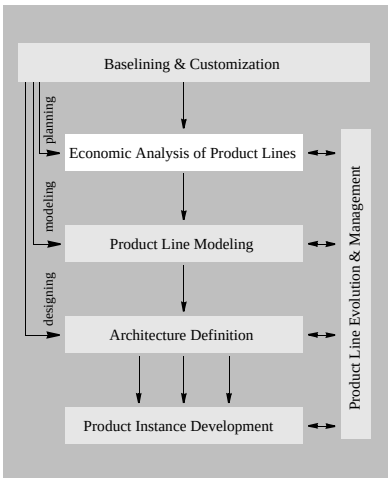
Vorgehen:
- Modell der Domäne erstellen
- Zukünftige Entwicklung der Domäne (!) abschätzen
- DSSA oder Referenzarchitektur entwickeln
- Einzelsysteme als Spezialfall dieser Referenzarchitektur „instanziieren"

Ansatz des IESE: PuLSE – **P**rod**u**ct **L**ine **S**oftware **E**ngineering

PuLSE – Product Line Software Engineering



PuLSE-ECO (Economic Analysis)



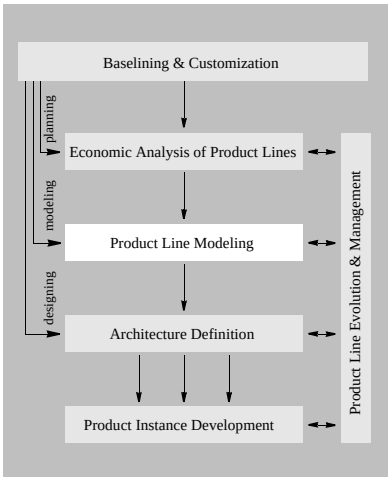
Ziel:

Identifiziere, beschreibe und begrenze eine Produktlinie so, daß der Bereich der Produktlinie die Ziele der Organisation möglichst gut abdeckt

→ Definiere und charakterisiere die Produkte aus der Linie, die erstellt werden sollen

→ Identifiziere die beste Vorgehensweise, um die Produkte zu entwickeln

PuLSE-CDA (Customizable Domain Analysis)



Ziel

Identifiziere, strukturiere und dokumentiere die Konzepte der Produktlinie und ihre Beziehungen untereinander

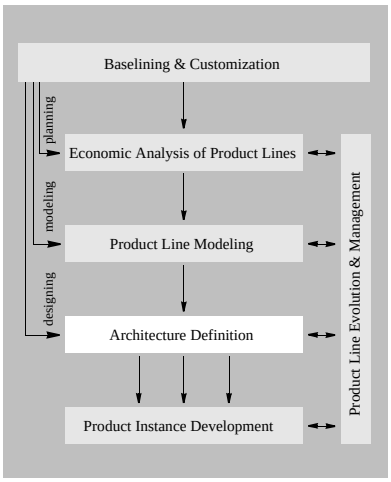
→ Benutze Dokumente (Work Products), die an die Domäne angepaßt sind

→ Benutze Szenarien, um das Domänenwissen zu erfragen

→ Konsolidiere das Wissen in einem expliziten Modell der Produktlinie

→ Verfeinere und validiere das Modell

PuLSE-DSSA (Domain-specific Software Architecture)



Ziel

- Entwickle eine Referenzarchitektur (DSSA), die gegenwärtige und potentielle zukünftige Produkte innerhalb der Produktlinie unterstützt
- Bewerte existierende Architekturkandidaten im Hinblick auf identifizierte Anforderungen

→ Entwickle inkrementell eine Referenzarchitektur
→ Benutze domänenspezifische Szenarien, um die Architektur zu validieren
→ Benutze domänenunabhängige Szenarien, um die strukturelle Qualität der Architektur zu bewerten

Aufgaben

- Entwickeln von Werkzeugen:
  - Glossar, verschiedene Formen von Dokumenten (Work Products)
  - Vernetzung, Referenzen
  - Durchgängige Verfolgbarkeit von Abhängigkeiten zwischen Anforderungen und Code (Traceability)
- Moderierte Gruppendiskussion zwischen „Experten” der Domäne
- Java

Literatur

GarlanShaw93	D. Garlan, M. Shaw, An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering Vol. 1, Singapur: World Scientific Publishing, 1993
GroschEmmelmann90	J. Grosch, H. Emmelmann, A Toolbox for Compiler Construction, CC’90: Lecture Notes in Computer Science, Vol. 477, Springer Verlag, Berlin, 1990
KazmanBass94	R. Kazman, L. Bass, Towards Deriving Software Architectures From Quality Attributes, CMU/SEI-94-TR-10, 1994
Knauber97	P. Knauber, Ein System für die Konstruktion objektorientierter Übersetzer, Dissertation Fachbereich Informatik, Universität Kaiserslautern, 1997
Kruchten95	P. B. Kruchten, The 4+1 View Model of Architecture, IEEE Software, 1995
Luckham et al.93	D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Verz, D. Bryan, W. Mann, Specification and Analysis of System Architecture Using Rapide, Stanford Technical Report, 1993
Trimble94	J. Trimble (ed.), STARS Program History 1983-1993, Version 1.1, available from the STARS Program Office
Schmitz et al.	L. Schmitz, F. Schulter, J. van Laak, SIC ‘95 Report and User Manual, Technischer Report Nr. 9501 Universität der Bundeswehr München, 1995
IESE/ISE-Web	http://www.iese.fhg.de/Competences/ISE/PLA