CASE

Motivation

- Ziele
  - Unterstützung der SW-Entwicklung durch Werkzeuge für Konstruktion, Analyse und Management
  - Effizienz-/Produktivitätsgewinn + Qualitätsverbesserung
- je reifer Entwicklungsprozesse sind, umso effektiver die Unterstützung durch Werkzeuge
  - einzelne Aktivitäten sind dann klar definiert
  - Ein-/Ausgabeartefakte sind spezifiziert

Computer Aided Software Engineering

Definitionen

- **CASE** befasst sich mit allen computerunterstützten Hilfsmitteln, die dazu beitragen, die SW-Produktivität und -Qualität zu verbessern sowie das SW-Management zu erleichtern (*Hinweis: „S" kann auch für System stehen*)
- **CASE-Werkzeug** ist ein Software-Produkt, das mind. eine einzelne bei der SW-Erstellung benötigte Aktivität unterstützt (automatisiert)
- **CASE-Workbench** besteht aus einer geringen Anzahl an CASE-Tools mit oft geringer Integration
- **CASE-Plattform/-Framework** stellt einen Rahmen für die Integration von CASE-Werkzeugen in eine CASE-Umgebung (SW-Entwicklungsumgebung) dar

Allgemeine Ziele

- Erhöhung der Produktivität
- Verbesserung der Qualität
- Erleichterung des Managements

Rahmenbedingungen

- CASE darf Entwickler nicht belasten, sondern soll Routine-Aufgaben abnehmen/erleichtern → Konzentration auf kreative Arbeiten
- Achtung: reine Einführung von Werkzeugen führt nicht zur Verbesserung der SW-Entwicklung („a fool with a tool is still a fool") → Einführung von Methoden mit *sinnvoller* Werkzeugunterstützung

Detailziele

- technische Ziele
  - Automatisierung von Entwicklungsaktivitäten
  - Methoden zur Konsistenz- und Redundanzprüfungen
  - Qualität der Dokumentation verbessern
  - Wiederverwendbarkeit erleichtern
  - Verwaltung von Konfigurationen und Änderungen
  - Messen und Bewerten
- wirtschaftliche Ziele
  - Effizienz erhöhen
  - Produkte schneller entwickeln (Time-to-Market)
  - Qualität der Produkte erhöhen

  - Wartungsaufwand reduzieren
  - Personenabhängigkeit verringern
- organisatorische Ziele
  - Unterstützung des gewählten Prozesses
  - Verbesserung/Präzisierung der Entwicklungsmethoden
  - Erhöhung der Standardisierung
  - jederzeitige Information über den Soll-/Ist-Stand (Messungen/Planungen)
  - Anpassung an geänderte Rahmenbedingungen

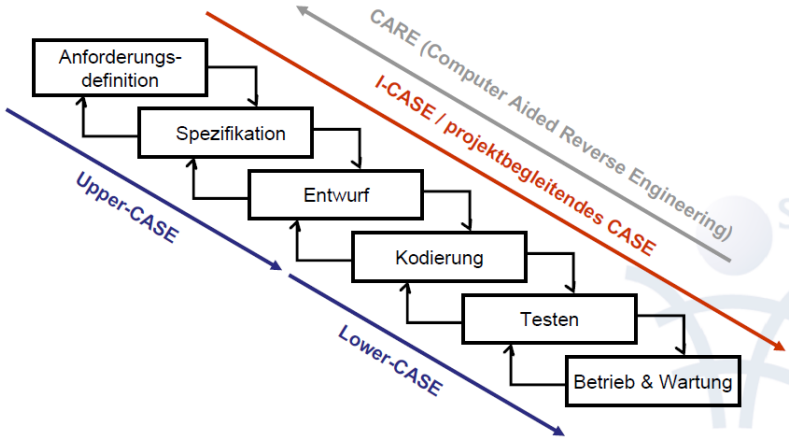
Klassifikation von CASE

Übersicht über verschiedene Perspektiven der Klassifikation

- Prozessperspektive: Einordnung bzgl. der Prozessschritte oder -phasen, die unterstützt werden
- Funktionale Perspektive: Einordnung bzgl. der unterstützten Art der Funktion bzw. Entwicklungs-aktivität
- Integrationsperspektive: Einordnung nach dem Grad der Integration (Umfang der unterstützten Prozessschritte/-fragmente)

Prozessperspektive

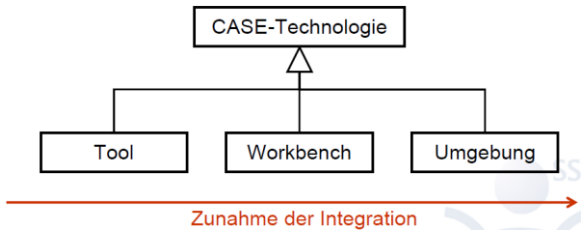
- je nach Schwerpunkt der Unterstützung existieren verschiedene Kategorien



Funktionale Perspektive (& Bezug zur Prozessperspektive)

- Codierung (→ Lower-CASE)
  - Eingabe (Editor), Übersetzung (Compiler)
- analytische Qualitätssicherung (→ ideal: I-CASE)
  - Test, Debugging, statische/dynamische Analyse, formale Verifikation
- Modellierung/Spezifikation (→ Upper-Case)
  - Zustandsautomaten, Funktionen, objektorientierte Modelle (UML, SDL)
- Requirements-Engineering und -Management (→ Upper-CASE)
- Prototyping/Simulation (→ Upper-CASE)
- Code-Generierung (→ Übergang Upper- zu Lower-CASE)
- Reverse Engineering (→ CARE)
- Versions- und Konfigurationsverwaltung, Änderungsmanagement (→ I-CASE)
- Planung (→ I-CASE)
- Dokumentation (→ ideal: I-CASE)

Integrationsperspektive



Werkzeugentwicklung

Motivation

- Entwicklung von CASE-Tools ist SW-Entwicklung
  - also quasi eine spezielle Anwendungsdomäne für das SE → angepasste Lösungen sinnvoll
- historisch
  - traditionelle Entwicklungstechniken (Implementierung „von Hand“)
  - Werkzeuge zur Generierung von Werkzeugteilen (z.B. Einlesen der Eingabedokumente, GUI)
- aktuell
  - modellbasierte CASE-Tool-Entwicklung
  - Generierung von Werkzeugen bzw. Werkzeugteilen aus Modellen

traditionelle CASE-Werkzeugentwicklung

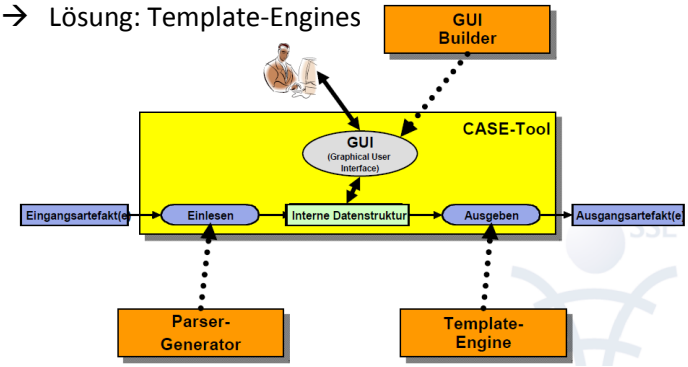
typische Probleme und Lösungen

- Problem: Einlesen von Eingangsartefakten
  - Eingangsartefakte sind häufig Textdokumente (Code, Konfigurationsdateien, Austauschformate für Modelle wie XMI)
  - Eingangsartefakte haben typischerweise ein vorgegebenes Format/Struktur
  - Ergebnis des Einlesens ist interne Repräsentation (Datenstruktur) für die Bearbeitung durch ein Werkzeug

  → Lösung: Parser-Generator
- Problem: Darstellung der Daten und Reaktion auf Benutzereingaben
  - Visualisierung der internen Datenstrukturen des Werkzeugs
  - Möglichkeit für (grafische) Benutzereingaben und Modifikationen der internen Datenstrukturen

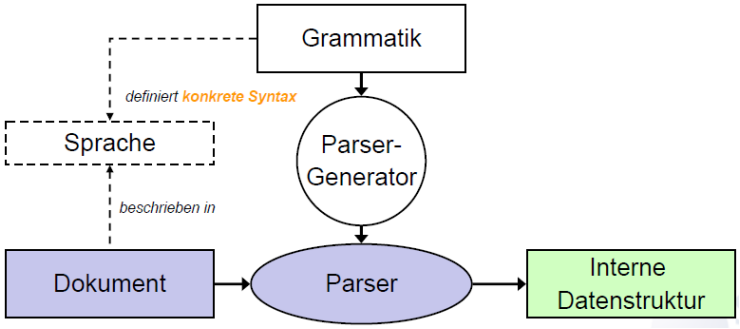
  → Lösung: GUI-Builder
- Problem: Erzeugung von Ausgangsartefakten
  - Überführung der internen Datenstruktur in „lesbare“ Ausgabe-/Textdokumente
  - Ausgangsformate haben typischerweise ein vorgegebenes Format/Struktur

  → Lösung: Template-Engines

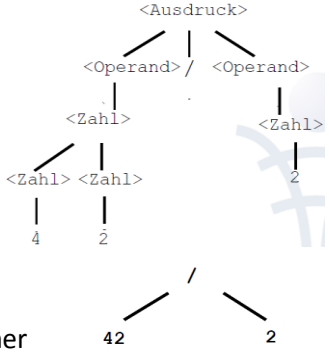


Parser-Generatoren

- automatisches Erstellen von Code zum
  - Einlesen von Artefakten mit gegebener Struktur (Syntax)
  - Überprüfung auf Einhaltung der Struktur
  - Überführung in eine interne Struktur
- Parser-Generatoren
  - bereits sehr lange verbreitet
  - Einsatz vor allem im Compiler-Bau („Drachenbuch“)
- allgemeiner Ansatz



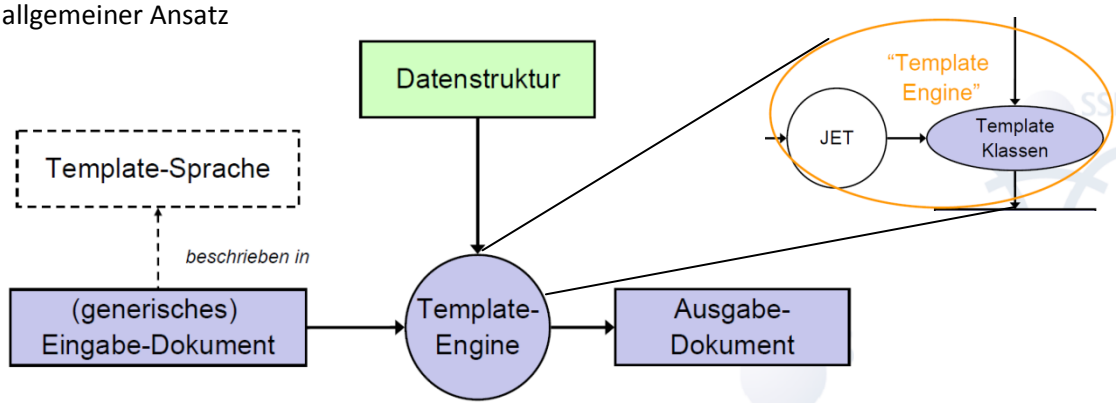
- Definition Grammatik
  - Grammatik G ist ein Tupel $G = (N, T, \Pi, Z)$, wobei N die Nichtterminalsymbole sind, T die Terminalsymbole, Π die Produktionsregeln und Z das Startsymbol
  - eine Sprache ist die Menge aller ableitbaren Wörter aus einer Grammatik
  - bestehen die Produktionsregeln nur aus Terminalsymbolen, heißt die Grammatik kontextfrei
- Parser-Vorgang
  1) lexikalische Analyse
     - Gruppierung einzelner Zeichen zu Tokens
     - Elimination von Whitespaces
  2) Syntaxanalyse
     - Gruppierung der Tokens zu grammatikalischen Einheiten
     - Abgrenzung zur lexikalischen Analyse ist z.T. willkürlich
     - Repräsentation als Parse-Baum bzw. AST (interne Datenstruktur)
  3) semantische Analyse
     - Überprüfung auf semantische Fehler (Type-Checking, korrekte Array-Indizierung)
- Parse-Baum
  - Wurzel des Baums ist immer Startsymbol Z
  - alle Blattknoten sind Terminalsymbole
  - alle internen Knoten sind Nichtterminalsymbole
  - reflektieren „Struktur“ der konkreten Syntax
- Abstract Syntax Tree (AST)
  - spiegeln abstrakte Syntax wieder
  - abstrahieren von konkreter Darstellung
  - machen Bearbeitung interne Datenstrukturen einfacher



- Beispiele existierender Tools
  - Unix → lex, yacc; Java → ANTLR; C#/Java → Grammatica; XML → SAX, DOM

Template-Engines

- Ziel
  - Erzeugung von Dokumenten
    - mit einer vorgegebenen Struktur
    - auf der Basis von Templates
  - Templates
    - Dokumente mit Platzhaltern für konkrete Daten
  - Template-Sprache
    - Beschreibung der Templates und insbesondere der Platzhalter
    - unterstützen Suchen & Ersetzen sowie komplexere Anfragen (z.B. Iterationen)
- Beispiele existierender Tools: JSP, Velocity, JET (Untermenge von JSP)
- allgemeiner Ansatz



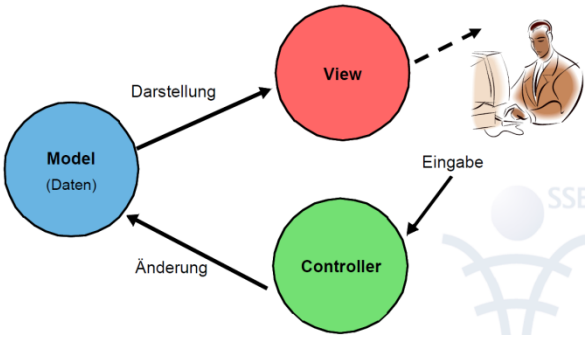
- Java Emitter Templates (JET)
  - Zwischenschritt: Generierung von Template-Klassen
    - Template-Klassen übernehmen das Erzeugen der eigentlichen Ausgabe zur Laufzeit
    - Effizienzgewinn durch Kompilation
  - Beispiel

```
<%@ jet package="src" imports="java.util.*" class="MyTemplate" %>
<% Person pp = (Person)argument; %>
<html> <body>
      Hello <%= pp.getName()%>!
      <ul>
            <% List li = (List)pp.getItems(); %>
            <% for (Iterator i = li.iterator(); i.hasNext();) { %>
            <li> <%=i.next().toString()%> </li>
            <% } %>
      </ul>
</body> </html>
```

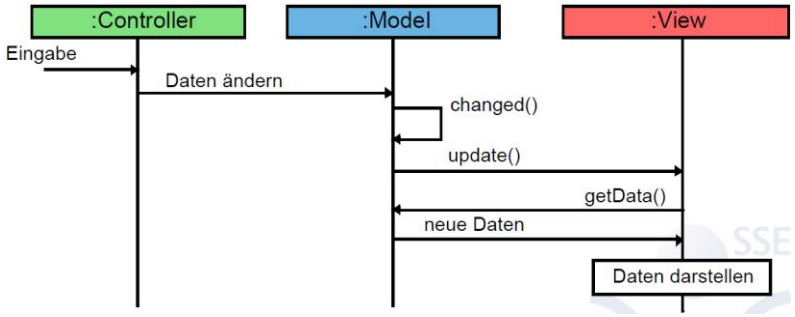
GUI-Builder

- Ziele
  - Erstellung von grafischen Benutzeroberflächen ohne „manuelle“ Programmierung
    - Eingabe durch Klicken
    - Auswahl der Widgets aus einer Palette

- Voraussetzung für sinnvollen Einsatz eines GUI-Builders
  - MVC-Pattern



  - Kommunikationsschema



Realisierung durch Observer-Pattern

Observable = Model

Observer = View

  - traditionelle Umsetzung in Java: Model + Component (= View + Controller)
  - beim SWT von Eclipse besteht wieder die saubere Trennung von MVC

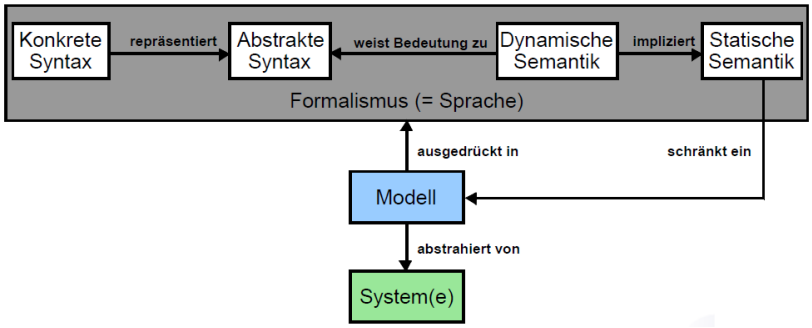
modellbasierte CASE-Werkzeugentwicklung

Motivation

- Beobachtung
  - viele CASE-Tools dienen der Bearbeitung von Modellen, die in einer spezifischen Sprache ausgedrückt sind
- Problem
  - Wie kann man solche modellbasierten CASE-Tools erstellen?
- Lösung
  - man nutzt Modelle zur Erstellung von modellbasierten CASE-Tools
  - Ausgangspunkt ist Modell der Sprache = Meta-Modell
  - → Meta-Modellierung als Grundlage

Grundlage: Meta-Modellierung

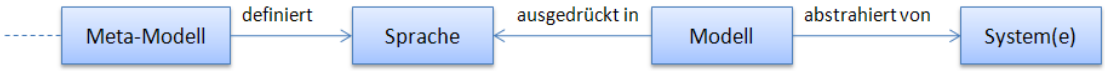
- Grundlagen: System, Modell, Formalismus



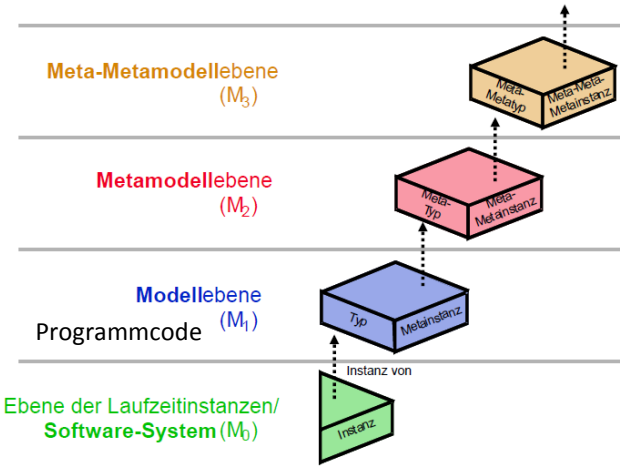
  - abstrakte Syntax (grunds. Sprachelemente): concepts from which models are created
    - „Zustand“

- konkrete Syntax (Repräsentation): concrete rendering of these concepts
  - ◯
- (dynamische) Semantik: description of a model's meaning
  - Menge aller Variablenbelegungen
- Well-formedness Rules (statische Semantik): rules for applying the concepts
  - exakt 1 Startzustand

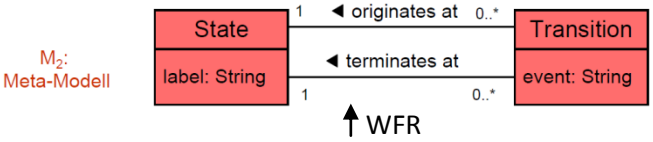
- Prinzip
  - ein Meta-Modell definiert die Sprache, mit welcher Modelle beschrieben werden
    - typischerweise definiert ein Meta-Modell die abstrakte-Syntax + WFRs



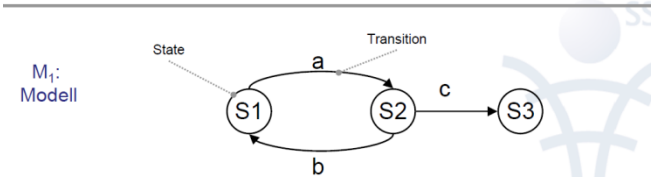
  - ein Meta-Meta-Modell definiert die Sprache zur Beschreibung von Meta-Modellen
- Modellhierarchie



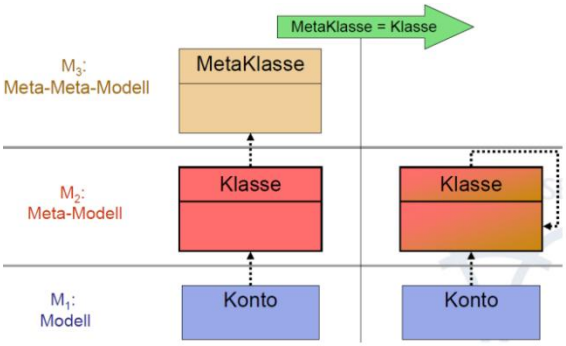
- Beispiel: Zustandsautomat



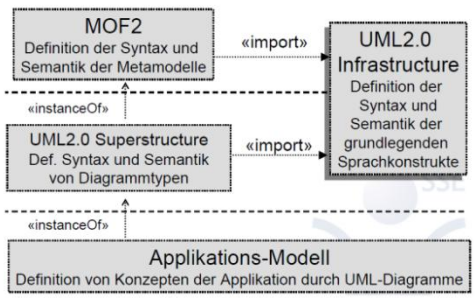
Aussehen und dyn. Semantik fehlen, Transition könnte auch xyz heißen



- Terminierung der Modellhierarchie
  - Alternative 1: andere Sprache für M_n → notwendig, um M_n zu verstehen (z.B. natür. Sprache)
  - Alternative 2: reflexiv: $M_{n+1} = M_n$ → typischer Ansatz für MetaCASE (Tools)

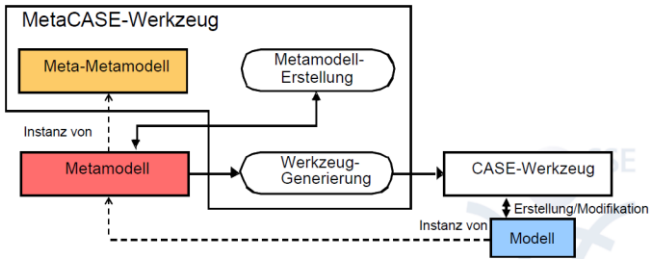


in der UML:



Meta-CASE-Entwicklung

- Meta-CASE-Tool ist ein CASE-Tool zur Erstellung von modellbasierten CASE-Tools



  - Metamodelle werden als Instanzen eines fest-codierten Meta-Metamodells spezifiziert
  - Meta-Modell-Editor ist Teil des MetaCASE-Tools
- Klassifikation

Klasse	Meta-Modell	Ausgabeartefakte
CASE einfaches, nicht anpassbares Modellierungswerkzeug	festgelegt	festgelegt
A-CASE anpassbares Modellierungswerkzeug (sehr verbreitet)	festgelegt	anpassbar
MetaCASE Metamodellierungswerkzeug mit festgelegter Abbildung	anpassbar	festgelegt (z.B. Standard-berichtsgeneratoren)
A-MetaCASE anpassbares Metamodellierungswerkzeug	anpassbar	anpassbar

- Abgrenzung
  - Code-Generierung in UML-Tools und MDA (Model Driven Architecture) der OMG
    - dient vornehmlich der Generierung „normaler“ Applikationen
    - Beispiele: OmondoUML, Rational Architect/Modeller
  - aber: Meta-Modell ist auch ein Modell
    - Konsequenz: modellbasierte Entwicklungswerkzeuge lassen sich prinzipiell auch zur Erstellung von CASE-Tools einsetzen (Modell = Meta-Modell)
    - nicht speziell dafür ausgelegt, daher Meta-CASE sinnvoller

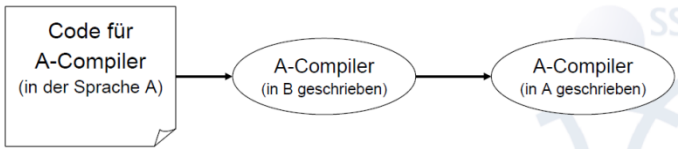
Beispiele für Meta-CASE-Tools

- DOME (Domain Modeling Environment): Honeywell
- GME (The Generic Modeling Environment): ISIS, Vanderbilt University
- MetaEdit+ und MethodWorkbench: MetaCASE
- Eclipse EMF (Eclipse Modeling Framework) & GMF (Graphical Modeling Framework): Eclipse.org
- Vergleich der CASE-Tools

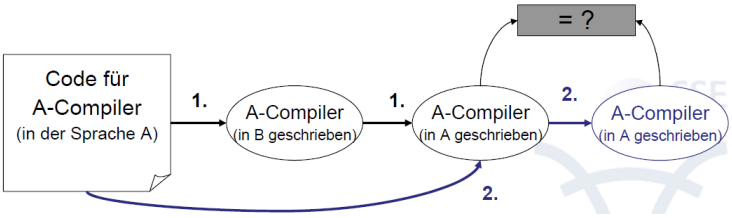
Eigenschaft	Werkzeug			
	DOME	GME	MetaEdit+	EMF/GMF
Notation (Meta-Meta-Modell)	Variante von Coad- Yourdon OOA: Nodes, Connectors	Variante der UML-Klassendia-gramme: Atoms, Connections	textueller EER-Dialekt: Objects, Relations, Roles, Ports	Ecore (Teilmenge von MOF)
Transforma-tionssprache	Data-Flow-Sprache und Scheme-Dialekt	Script-Sprache	keine integrierte	Java
Dokument-generation	Transformations-sprache; template-basiert	musterbasiert	eigene Report-Generierungsspra che	Transformations-sprache (Java); template-basiert (JET)
Zugriff auf Repository	interne Erstellung eines Data-Dictionarys möglich	externer Zugriff über COM	externer Zugriff über SOAP (Web-Services)	Direkt; über Editor-Klassen
Austausch-format	bel. Export (über Dokumentgen.)	Ein- und Ausgabe von XML	vordefinierter Export nach XML	XMI

Bootstrapping für Meta-CASE-Tools

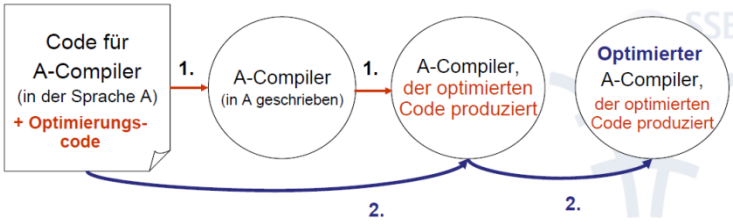
- Ziel
  - Entwicklung eines Compilers für die Sprache A (A-Compiler)
  - aber: Programmcode für den A-Compiler soll in A programmiert werden, da
    - die A-Sprache sehr mächtig ist oder neue Konzepte wie OO oder AOP besitzt
    - Compiler-Entwickler nur die Sprache A kennen müssen
- Problem
  - zum Kompilieren des A-Compilers, der in A programmiert wurde, braucht man einen A-Compiler
- Lösungen
  1) andere Sprache: A-Compiler wird zunächst in Sprache B geschrieben, dann kann man damit A-Compiler (in Sprache A) kompilieren



  2) Cross-Compilation: es existiert ein A-Compiler auf einer anderen Plattform P
     - Änderungen der Ausgabe des existierenden Compilers in Maschinencode für Zielplattform Q
     - Kompilation
     - Kopieren des Bytecodes von Q nach P
  3) manuelle Übersetzung: Code des A-Compilers wird von Hand nach Assembler übertragen
  4) Teilmenge: erste Versionen des A-Compilers werden für eine Teilmenge der Sprache A geschrieben (inkrementelles Vorgehen); z.B. rudimentäre Zuweisungen, goto, …
- Vorteile
  - Nutzen des Bootstrappings zum „Testen des Compilers“
    - A-Compiler (in A geschrieben) muss in der Lage sein, denselben Maschinencode zu produzieren wie der A-Compiler (in B geschrieben)



  - Nutzen des Bootstrapping zur „Verbesserung des Compilers“
    - eine Verbesserung des Compiler-Programmcodes führt nicht nur zur Verbesserung der allgemeinen Applikationen, sondern auch zu einer Verbesserung des Compilers selbst



- MetaCASE-Tool-Entwicklung
  - Ausnutzen der reflexiven Terminierung der Meta-Hierarchie ($M_{n+1} = M_n$)
    $\rightarrow M_4 = M_3 \rightarrow$ Meta-Meta-Metamodell = Meta-Metamodell
  - Bootstrapping mit traditionell entwickeltem Meta-CASE-Tool → entspricht Lösung 1

Werkzeugintegration

Motivation

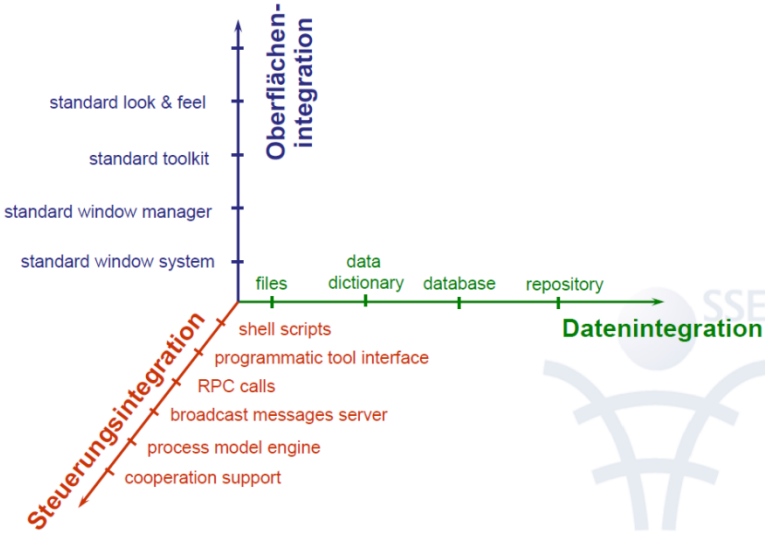
- Probleme bei Werkzeugen, die unabhängig voneinander entwickelt wurden
  - Kombination der einzelnen Tools zu Workbenches oder Umgebungen (Upper-/Lower-/I-CASE) oft schwierig (z.B. bei Verwendung verschiedener Methoden)
  - fehlende Zwischenschritte oder Überlappung von Funktionen/Diensten (z.B. jeweils eigene Versionsverwaltung)
  - fehlende standardisierte Prozeduren und Mechanismen für die Übertragung von Daten von einem zum anderen Werkzeug und die Synchronisation zwischen den Tools (z.B. PDF/Word)
  - mangelnde Übersicht über den Gesamtfortschritt der Aktivitäten und die Zwischenschritte von Artefakten, welche produziert werden
  - hoher Aufwand in der Systemadministration (z.B. Installation, Betrieb)

Prozessmodellierung

Klassifikation der Ansätze zur Werkzeugintegration

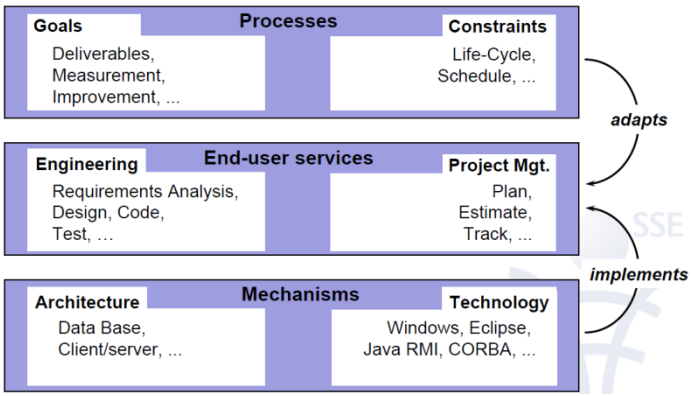
Wassermann-Modell

- 5 Dimensionen der Integration
  - Plattformintegration
    - Werkzeuge laufen auf derselben Plattform (HW und OS)
  - Oberflächenintegration
    - Werkzeuge besitzen dasselbe Look-and-Feel
  - Datenintegration
    - Daten lassen sich zwischen den Werkzeugen austauschen (z.B. XML)
  - Steuerungsintegration
    - Dienste eines anderen Werkzeuges lassen sich aufrufen
    - Tool kann von anderem Werkzeug über Ereignisse benachrichtigt werden
  - Prozessintegration
    - Zusammenarbeit der Tools im Rahmen eines definierten Prozesses
    - CASE-Umgebung kennt Prozessmodell
- Ausprägungen



- Problem 1: Dimensionen sind nicht unbedingt orthogonal
  - Dimensionen können daher nicht immer unabhängig voneinander betrachtet werden
  - Bsp Daten- und Steuerungsintegration: Einschränkung des Datenzugriffs ist eine Form der Steuerung, Nachrichtenaustausch zw. Werkzeugen beinhaltet meist auch Daten
- Problem 2: eine Zunahme entlang der Achsen impliziert nicht unbedingt Verbesserung
  - Techniken werden zwar ausgefeilter, aber nicht notwendigerweise semantisch gehaltvoller (so kann eine XML-File mehr Semantik haben als eine einzelne Tabelle)

Brown-Modell

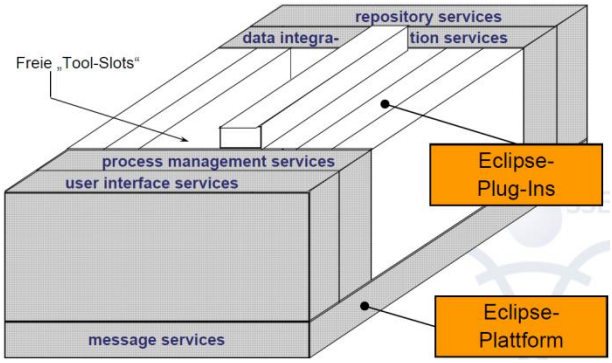


- End-User Services
  - abstrakte Beschreibung der Funktionalität (Dienste)
  - Integration beschreibt die Beziehung dieser Dienste (auf abstrakter Ebene)
    - z.B. Zusammenhang zwischen Versionsverwaltung und Datenhaltung
  - „What does this environment do?“
- Mechanisms
  - technologische Umsetzung und Architektur zur Realisierung der Services
  - Integration bezieht sich auf Implementierungs-Aspekte
    - z.B. Austausch von Nachrichten und Daten über CORBA-Middleware
  - „What components are used und how does this environment integrate the various component?“
- Processes
  - Einbettung der Services in den Kontext des Gesamtprozesses
  - Integration beschreibt die mögliche Abfolge der Services (Servicebeschreibung)
    - z.B. Bug-Tracking- und Versionsverwaltungs-Services werden nach Auffinden/Korrigieren des Fehlers nötig
  - „Which processes does this environment support and how does the process influence the use of end-user services?“

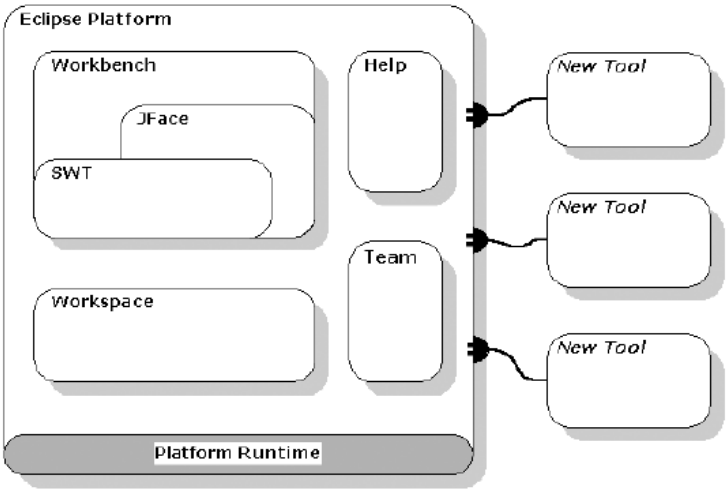
NIST/ECMA-Referenzmodell

- Grundidee
  - standardisiertes Grundgerüst (Framework) zum Vergleich von CASE-Umgebungen (abstrahiert von der konkreten Architektur der CASE-Umgebungen)
  - Framework beinhaltet eine Menge tool-unabhängiger (grundlegender) Dienste, welche für die Realisierung einer CASE-Umgebung benötigt werden
- Synonyme
  - integration plattform = infrastructe = common service

- „Toaster Model“



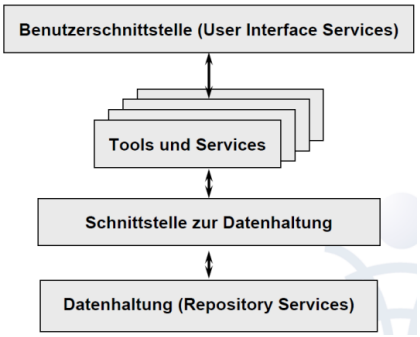
  - Repository Services
    - Verwaltung von Entitäten und deren Beziehungen (inkl. evtl. Verteilung der Objekte)
  - Data-Integration Services
    - höhere semantische Strukturierung der Daten (z.B. mittels Metamodell)
    - Versions- und Konfigurationsmanagement
    - Datenaustausch mit externen CASE-Tools
  - Process-Management Services
    - Verwaltung und Ausführung zu erledigender Aufgaben
    - Aufrufsequenzen der einzelnen Tools
  - Message Services
    - Kommunikation zwischen den Tools, den Services sowie den Tools und den Services
    - Werkzeug-Registrierung erlaubt es, sich für den Empfang spezieller Multicast-Nachrichten zu registrieren
  - User-Interface Services
    - einheitliche Benutzerschnittstelle (Metaphern, Views, Controller)
    - trennt Bedienung von Funktionalität
    - übernimmt OSF’s Layered Reference Model für UI
  - Bezug zwischen Eclipse und Toaster Model



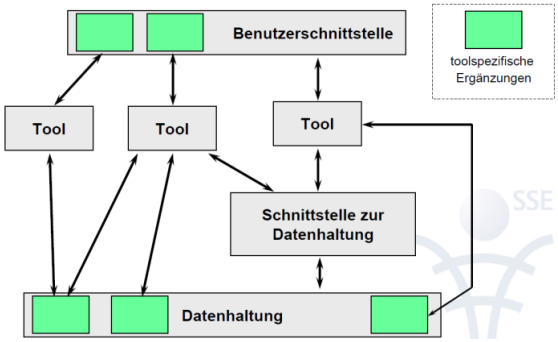
    - Workspace = Repository Services
    - Workbench (inkl. JFace & SWT) = User Interface Services
    - Team (Support), EMF (als ext. Plugin) = Data Integration Services (Versionsverwaltung)
    - Platform Runtime = Message Services
    - (nur rudimentär) = Process Management Services

- Werkzeugintegration

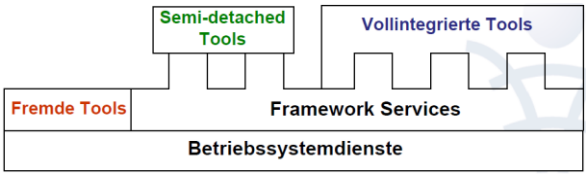
ideale Architektur



reale Architektur (Praxis)



  - vollintegrierte Tools
    - gesamte Daten werden durch Common Services verwaltet und innerhalb des Frameworks integriert
  - teil-abgetrennte Tools (semi-detached tools)
    - verwalten eigene Datenstrukturen, aber die Files mit dem Inhalt der Datenstrukturen werden über Common Services verwaltet
  - fremde Werkzeuge
    - laufen auf derselben Maschine (Netzwerk)
    - benutzen nur Betriebssystemdienste für die Integration/Kommunikation
    - Datenaustausch mit der Tool-Umgebung erfolgt über Data Interchange Services



Verständnis der 5 Dimensionen der Integration

Plattformintegration

- Ziel der Integration
  - „Verstecken“ der Verteilung der einzelnen SW-Komponenten (Tools)
  - Ausgleich der Heterogenität der HW und der Betriebssysteme
- Benötigte Services
  - verteiltes Dateisystem
    - Standardfunktionen zur Dateiverwaltung
    - Speicherung auf Servern, die vom Netzwerk zugänglich sind
  - Kommunikations-Services
    - Unterstützung der Interprozess-Kommunikation zw. den Computern des Netzwerks
  - Prozessmanagement-Services
    - Erzeugen, Starten, Beenden von Prozessen (Programmen) auf Rechnern des Netzwerks
  - Druck-Services
    - Ausdruck aus dem Netzwerk
- Beispiel Eclipse
  - Begriffe
    - Eclipse-Plattform = CASE-Framework
    - Eclipse-Plug-In = CASE-Tool
    - Eclipse-Plattform + Plug-Ins = CASE-Umgebung

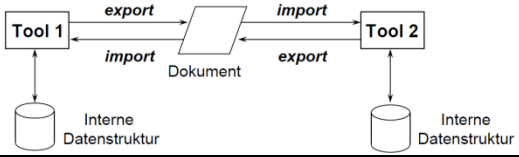
- Plattformintegration
  - Implementierungssprache von Eclipse ist Java → Plattformunabhängigkeit
  - WebDAV-Protokoll erlaubt die Integration von Tools über verschieden OS hinweg

Oberflächenintegration

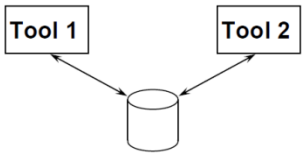
- Ziel der Integration
  - Verringerung des Lernaufwands für ein neues Tool (durch gleiches Erscheinungsbild aller Tools, gleiche Metaphern, gleiche Kommandos für gleiche Aktivitäten)
- 3 Ebenen der Integration
  - Integration auf Ebene des Windowing-Systems
    - Tools besitzen identische Fenster oder auch identische Widgets
  - Kommando-Integration
    - vergleichbare Befehle besitzen denselben Namen, sind an derselben Stelle im Menü, besitzen dieselbe Darstellung
  - Interaktions-Integration
    - Manipulation grafischer Elemente ist analog/identisch (z.B. Lasso in MS Office)
- Beispiel Eclipse
  - Problem: Abwägen zwischen Plattformunabhängigkeit und OS-Integration
    - Java AWT: native Implementierung, nur geringe Schnittmenge für alle OS
    - JFC/Swing: Emulation von Widgets, Performance-Einbußen
  - Lösung in Eclipse: SWT (Standard Widget Toolkit)
    - native Implementierung wenn möglich, sonst Emulation
    - erlaubt stets Look & Feel des Host-OS
    - API unabhängig vom OS

Datenintegration

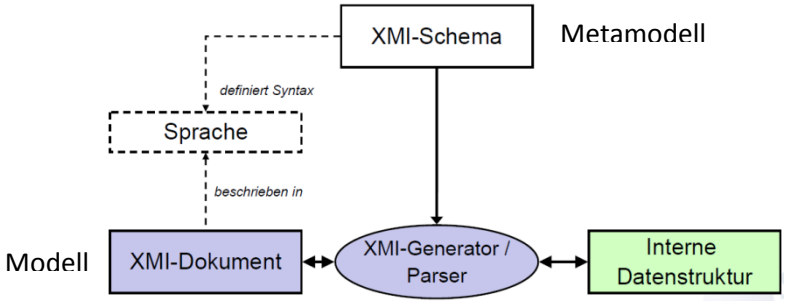
- Ziel der Integration
  - unterschiedliche Werkzeuge können Daten austauschen bzw. gemeinsam darauf zugreifen
- Hauptfragen
  - Austausch von Daten
    - Welche Daten werden gespeichert, persistent oder nicht-persistent?
    - Wo werden die Daten gespeichert?
    - Wie erfolgt der Zugriff auf die Daten?
  - semantische Mechanismen
    - Inwiefern besitzen die Tools ein gemeinsames Verständnis der Daten?
    - Welche Mechanismen werden zur Unterstützung des gemeinsamen Verständnisses verwendet?
  - Konsistenz der Daten
    - Wie wird die Konsistenz der Daten gewährleistet?
    - Wie redundanzarm sind die Daten?
- 2 prinzipielle Ansätze
  - Import/Export (z.B. SVN)



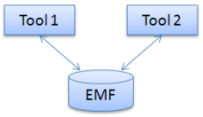
- gemeinsamer Datenspeicher mit einheitlichem, zentralisiertem (DB-)Schema



- semantische Mechanismen
  - Beschreibung von Meta-Daten
    - Spezifikation der Bedeutung der Daten
  - Ansatz 1: gemeinsames Datenformat
    - Bedeutung ist durch Struktur der Daten festgelegt
    - Anwendung hauptsächlich für den Import/Export-Ansatz
  - Ansatz 2: gemeinsames Schema (Modell)
    - Bedeutung ist durch Beziehungen zwischen den Daten und durch Datentypen festgelegt
    - Anwendung hauptsächlich für den „gemeinsamen Datenspeicher“-Ansatz
- gemeinsames Schema
  - Schema beschreibt Typen von Daten (Objekte), Eigenschaften dieser Objekte (Attribute), Beziehungen zwischen den Objekten (Relationen) und Zugriff auf Objekte
  - Problem: Entwicklung eines gemeinsamen Standards erfordert einen hohen Abstimmungsaufwand zwischen den einzelnen Tool-Entwicklern
  - OMG Standards zur Lösung dieser Probleme (modellbasierte SW-Entwicklung, Daten = Modell, Schema = Meta-Modell)
    - MOF (Meta-Object Facility): Standard Meta-Modellierungssprache
    - XMI (XML Metadata Interchang): XML-Dialekt für Import/Export von OMG-Modellen, Metamodell der Sprache ist in MOF spezifiziert



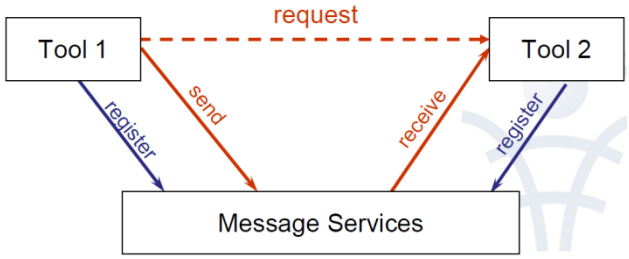
    - UML Profile: spezielle Metamodelle für spezifische Anwendungsgebiete
- Beispiel Eclipse
  - Eclipse-Plattform („Boardmittel“)
    - Resources = Dateien, Verzeichnisse, …
    - einfache Form der Integration: Speicherung der Daten in Files, File-Access durch Tools
    - Unterstützung durch Eclipse-Framework: Resource-Manager erlaubt den Zugriff auf Resources über spezifische API (z.B. Notifications bei Änderungen)
  - Eclipse Modeling Framework (EMF)



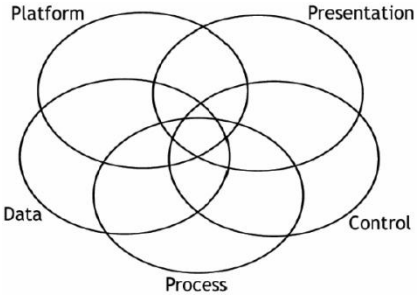
    - Modellierung der Schemata mit Ecore (⊂ MOF)
    - Generierung der Java-Klassen: Ecore → Java
    - erlaubt semantisch höherwertige Form der Integration im Vergleich zu reinen Files (gemeinsamer Zugriff auf Daten durch Extension Points anderer Plug-Ins oder durch Austausch von XMI)

Steuerungsintegration

- Ziel der Integration
  - Anfrage der Dienste anderer Tools (request)
  - Benachrichtigung über Ereignisse anderer Tools (notify)
  - beinhaltet auch Integration mit (und zwischen) den Common Services der Plattform
    - Aufruf eines Compilers aus einem Editor heraus (request)
    - Einblendung der Compiler-Fehlermeldung in den Editor (notify)
    - Benachrichtigung über Änderung eines Files auf der Platte (Common Services)
- prinzipieller Ansatz
  - Steuerungsnachrichten zum Versand von Aufgaben und Benachrichtigungen
  - Aufgabe des Message Services
    - Tools registrieren sich für den Nachrichtenaustausch



- Steuerungsnachrichten
  - Tool-Umgebung
    - abstrakte Sicht: Menge von Aktionen, welche durch die Tools zur Verfügung gestellt werden
    - Koordination der Tools erfolgt durch den Austausch von Steuerungsnachrichten, die mit Aktionen assoziiert sind
  - Arten von Steuerungssignalen
    - Bekanntgabe durchgeführter Aktionen (notify)
    - Anfrage durchzuführender Aktionen (request)
- Bezug zu Datenintegration
  - Steuerungsintegration erfordert auch Austausch von Daten → Überlappung der Dimensionen
  - Ebenen der Steuerungsintegration haben engen Bezug zu Ebenen in der Datenintegration
  - verbessertes Dimensionenmodell

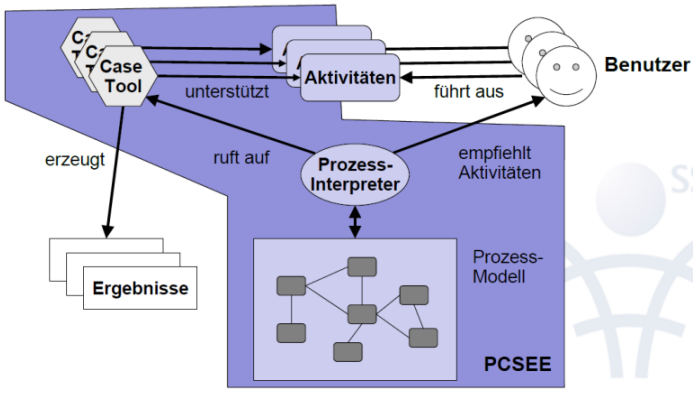


- Integrationstiefen
  - Übertragungsebene: Nachrichten sind einfache Text-Strings
  - lexikalische Ebene: Dateneinheiten (Tokens) sind festgelegt
  - syntaktische Ebene: Syntax der Nachricht ist festgelegt
  - semantische Ebene: Bedeutung der Token (bzw. der gesamten Nachricht) ist festgelegt
  - Prozess-/Methodenebene: Tools besitzen Kenntnis über den Entwicklungsprozess

- Beispiel Eclipse
  - Rolle von Java
    - alle Plug-Ins sind in Java realisiert → vereinfach die Steuerungsintegration (Nachrichtenaustausch kann durch Java-Methoden erfolgen, bei Bedarf auch über JavaRMI)
  - Mechanismen der Steuerungsintegration
    - Event/Listener-Mechanismus von Java: Plug-Ins können sich u.a. registrieren um eine Zusammenfassung der Änderungen der Ressourcen zu erhalten , Integrationstiefe: mind. syntaktisch (Datentypen), i.d.R. semantisch
    - Extension Points und Extensions: Schnittstellen von Plug-Ins, Integrationstiefe: mind. syntaktisch (Datentypen), i.d.R. semantisch
    - Perspectives: Zusammenstellung von Views, Editoren und Menüs für eine gewissen Aufgabe, Integrationstiefe: Prozess-/Methodenebene

Prozessintegration

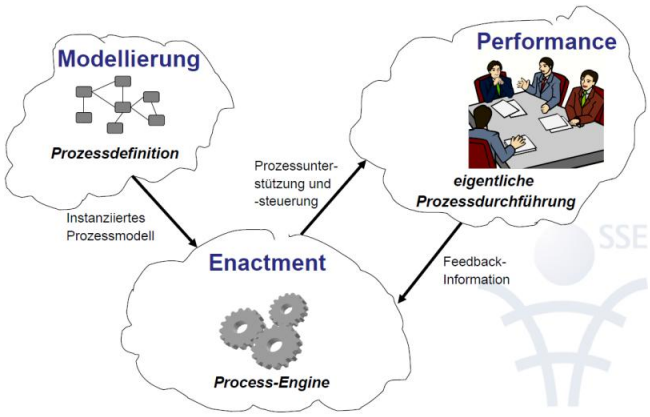
- Ziele
  - jedes Tool verkörpert Annahmen über den Entwicklungsprozess
    - Texteditor (sehr wenige Annahmen), Code-Generator (sehr viele Annahmen)
  - Ziele der Integration
    - CASE-Umgebung kann die Entwickler aktiv bei der Durchführung des Entwicklungsprozesses unterstützen (Hinweise wann welche Aktivität durchgeführt werden sollte, automatischer Aufruf der jeweiligen einzusetzenden Tools)
    - Umgebung ist aktiv an der Festlegung des Ablaufs von Aktivitäten beteiligt
    - Prozess ist nicht fest kodiert, sondern flexibel festzulegen → Prozessmodell ist Eingabeparameter
  - PCSEE (Process-centered Software Engineering Environment)
    - CASE-Umgebung, welche obige Ziele erreicht
- Voraussetzungen
  - CASE-Umgebung besitzt Kenntnis des Entwicklungsprozesses (Artefakte und Aktivitäten, deren Beziehungen, deren Einschränkungen und die jeweiligen Werkzeuge)
  - Realisierung eines PCSEE erfordert, dass Tool-Umgebung
    - Modell des Software-Prozesses besitzt
    - Prozess-Interpreter (Process Engine) zur Ausführung des Modells beinhaltet



- PCSEE – Einschränkungen
  - Flexibilität vs. Prozessunterstützung
    - je flexibler eine PCSEE ist, umso schwieriger wird die Unterstützung für einen konkreten Prozess (da keine Detailinformationen über die verwendeten Tools, etc. vorliegen)

- Mächtigkeit der Process-Engine
  - nur für stabile und Routine-Prozesse ist Steuerung des Prozesses durch eine Process-Engine realistisch
  - dynamische Prozesse mit vielen Ausnahmen lassen sich oft nicht detailliert beschreiben, um eine automatisierte Ausführung zu ermöglichen
- Einschränkungen durch detaillierte Prozessmodelle
  - eine detaillierte Prozessmodellierung (und PCSEE-Unterstützung) kann mehr einschränken als unterstützen (da im Detail alle Dokumente und Konsistenzbedingungen beschrieben und eingehalten werden müssen → kein sinnvolles CASE mehr)

Domänen eines Software-Prozesses

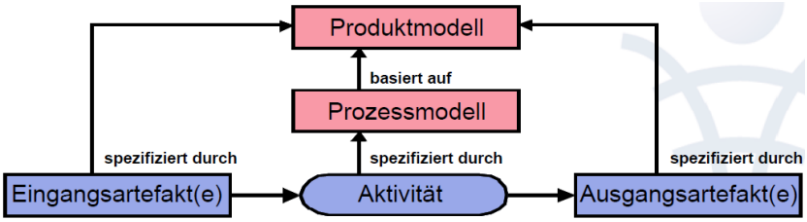


- Prozessdomäne Modellierung im PCSEE
  - Inhalt
    - Aktivitäten zur Definition und Wartung der SW-Prozessmodelle
  - Output
    - instanziiertes Prozessmodell (= Prozessplan)
    - Parameter zur Instanziierung: existierende Ressourcen, Zeitvorgaben, …
- Prozessdomäne Enactment im PCSEE
  - Inhalt
    - Interpretation und Ausführung eines instanziierten Prozessmodelle
  - Input
    - instanziiertes Prozessmodell (von Modellierung)
    - Feedback (von Performance)
  - Output
    - Informationen und Befehle zur Steuerung und Unterstützung der eigentlichen Prozessdurchführung (für Performance)
- Prozessdomäne Performance im PCSEE
  - Inhalt
    - Menge aller eigentlichen, durch Agenten (Menschen/Tools) durchgeführten Aktivitäten
  - Input
    - Informationen und Befehle zur Steuerung und Unterstützung der eigentlichen Prozessdurchführung (von Enactment)
    - möglich: Prozessmodell (von Modellierung), wenn z.B. ein RE-Ingenieur sein Verhalten entsprechend den festgehaltenen Ratschlägen im Prozessmodell anpasst
  - Output
    - Feedback-/Monitoring-Informationen (für Enactment)

Prozessmodellierung

Begriffe und Abgrenzung

- ein **Software-Prozess** entspricht der Durchführung einer SW-Entwicklung und ist gekennzeichnet durch
  - wann und von wem welche Aktivitäten durchgeführt werden
  - welche Artefakte bei der Durchführung der Aktivität verarbeitet/erzeugt werden
- ein **Software-Prozessmodell** dokumentiert SW-Prozesse durch Definition von
  - (alle) Arten von Aktivitäten die in der Entwicklung durchzuführen sind
  - Reihenfolge in der Aktivitäten durchzuführen sind (oft Vorgehensmodell genannt)
  - Methoden und Werkzeuge, die zur Unterstützung dieser Aktivitäten verwendet werden sollen
  - Arten von Artefakten, die durch die Ausführung der Aktivitäten erzeugt/verarbeitet werden (genauere Definition dann in einem Produkt-/Artefaktmodell)



- **Software-Prozessmodellierung** befasst sich mit der Definition, Verfeinerung und Erprobung von Prinzipien, Methoden, Techniken und Werkzeugen zur Unterstützung
  - der Kommunikation über SW-Entwicklungsprozesse
  - der Ablage von SW-Engineering-Prozessen basierend auf wiederverwendbaren Komponenten
  - der Analyse von Prozessen und dem Ziehen von Schlussfolgerungen
  - der Projektsteuerung und -kontrolle
  - der automatischen Unterstützung durch PCSEEs
- Warum Software-Prozess*modellierung*?
  - Beherrschung von Komplexität, Steigerung von Qualität und Produktivität, Wiederholbarkeit
  - Unterstützung der Prozessverbesserung sowie des Projekt- und Risikomanagements
  - rechnergestützte Anleitung während der Projektdurchführung
  - automatische Abwicklung von Prozessmodellen (Enactment)
  - Unterstützung der Analyse des Entwicklungsprozesses
  - Verbesserung der Kommunikation zwischen Projektmitgliedern
- ein **Prozess** besteht aus zwei Arten von **Prozesselementen**
  - **Subprozesse**, die wiederum als eigenständige Prozesse betrachtet werden können
  - **Prozessschritte**, die atomar sind und nicht weiter verfeinert werden können und somit eng zusammenhängende Tätigkeiten umfassen
  - rekursive Verfeinerung, so lange bis keine weiteren sinnvollen Strukturierungen mehr möglich sind aufgrund starker personeller und zeitlicher Kopplung der Tätigkeiten oder wenn keine assoziierten Produkte/Artefakte existieren
- ein Prozessschritt wird durch die folgenden **Attribute** (Eigenschaften) genauer beschrieben
  - Rollen (z.B. Tester)
  - anwendbare Prozess- und Produktstandards (z.B. Test-Driven-Development)
  - anwendbare Prozeduren, Methoden, Werkzeuge und Ressourcen (z.B. JUnit, 2 Tester/Tag)

  - Eingangskriterien (z.B. Testfall und „Artefact under Test“ fertig gestellt)
  - Eingangsartefakt (z.B. Testfall und „Artefact under Test“)
  - Produkt- und Prozessmaße, die gesammelt und verwendet werden sollen (z.B. Anzahl Fehlerwirkungen/Failures)
  - Verifikationspunkte (z.B. 80 % Testüberdeckung erreicht)
  - Ausgabeartefakte (z.B. Test-Protokoll)
  - Schnittstellen (z.B. Übergang zu Debugging)
  - Ausgangskriterien (z.B. „bar is green“)
- Prozesselemente
  - **Schnittstellen** zwischen Prozesselementen und zu externen Prozessen
  - **Reihenfolge** der Ausführung der Prozesselemente (Sequenz, Alternative, Schleife, Fork/Join)
  - **Abhängigkeiten** zwischen den Prozesselementen (z.B. wenn A begonnen hat, darf erst B starten)
- ein **Prozessmodell** ist die definierende Beschreibung eines Prozesses, d.h. es umfasst Prozesselemente und deren Beziehungen, außerdem (über Attribute der Prozessschritte und Beziehungen)
  - Input-/Outputprodukte (Artefakte)
  - Rollen und ihre Zuordnung
  - Zuordnung von Techniken, Methoden und Werkzeugen
- **Projektplan**
  - spezielle Sicht auf ein Prozessmodell (ist nach wie vor ein Prozessmodell)
  - basierend auf projektspezifischen Informationen (Ressourcen, Einschränkungen) angepasst, angereichert und instanziiert
- Eigenschaften eines SW-Prozesses
  - Entwicklung großer SW-Systeme ist ein komplexer Prozess
    - quantitative und qualitative Vielfalt von Aufgaben
    - große Anzahl an beteiligten Personen (Stakeholder)
  - Entwicklung komplexer Systeme erfordert
    - Prinzipien, Techniken, Methoden
    - Werkzeuge
    - SW-Entwicklungsprozess → Projektplan
  - es besteht eine direkte Abhängigkeit zwischen dem SW-Entwicklungsprozess und den Qualitätseigenschaften der SW
    - Prozessqualität beeinflusst die Produktqualität (richtige Balance finden zwischen starrer Vorgaben und totaler Freiheit)
  - Konzentration auf isolierte Methoden und Techniken des SW-Engineerings allein genügt nicht → Entwicklungsprozesse müssen als Ganzes betrachtet werden
- Geschäftsprozess
  - erfolgsrelevante grundlegende Unternehmenstätigkeiten, die zur Umsetzung der Unternehmensziele und Sicherung des Unternehmenserfolgs dienen
  - Unterteilung in Kernprozesse/strategische Prozesse und Supportprozesse
- SW-Prozessmodellierung vs. GP-Modellierung
  - das Produkt SW besitzt einige besondere Eigenschaften, das es von anderen Produktkategorien grundlegend unterscheidet: Komplexität, Änderbarkeit, Unsichtbarkeit, Art der Fertigung, Art der Zurverfügungstellung

- SW-Prozesse ähneln GPs, aber mit zusätzlichen Herausforderungen
  - sehr feine Granularität
  - Softness und Anpassbarkeit (Änderung/Evolution des Prozessmodells, Änderung des Prozesses während der Durchführung)
  - → starre Modellierung von vordefinierten Prozessen wie bei der GP-Modellierung ist bei der SW-Entwicklung i.d.R. nicht möglich

Prozessmodell-Klassifikation

Typen von Prozessmodellen

- Universal – U (auch „life-cycle“)
  - Waterfall, Spiral Model, Prototyping, incremental/iterative development, reusable software model, model-driven software development, agile development
- Worldy – W
  - beschreibt die prinzipielle Abfolge von Schritten
  - unterstützt den Entwickler (Guidelines)
  - Bsp. IBM Cleanroom, UP, XP (agile Methoden)
- Atomic – A
  - beschreibt Prozessschritte, die in einer systematischen und evtl. automatisierten Art und Weise durchgeführt werden können (Prozeduren, Algorithmen, …)

Abstraktionsebenen

- Generisch
  - organisationsweiter Standard/Rahmen
  - vereinfacht Training, Review und Tool-Support
- Angepasst (Tailored)
  - passend für spezifische Projekt-Charakteristiken
- Enacted
  - instanziiertes Prozessmodell (= Projektplan + Projekt-Status)

Einsatzweck

- deskriptive Prozessmodellierung
  - Erfassung und Beschreibung des tatsächlichen Prozesses zur Kommunikation & Analyse
  - Zweck: Verstehen, Kommunizieren, Messen, Führen
  - geeignet zur Prozessverbesserung
- präskriptive Prozessmodellierung
  - Konzeption und Beschreibung des gewünschten Ablaufs
  - geeignet für Anleitung von Prozessbeteiligten

Beschreibungsansatz

- ablauforientiert („operational“)
  - explizite Steuerung in welcher Reihenfolge Aktivitäten durchlaufen werden
  - Vorteil: Verständlichkeit
  - Nachteil: Flexibilität
- constraint-basiert („deklarativ“)
  - psychologische Beobachtungen zeigen: Entwickler sind in einer einengenden Umgebung nicht kreativ und produktiv

→ nur diejenigen Prozessabfolgen ausschließen, die sich als kontraproduktiv zu den Zielen des Unternehmensherausstellen
→ ergebnisorientiert (Akzeptanzkriterien für Produkte)
- Vorteil: psychologischen Erkenntnissen wird Rechnung getragen
- Nachteil: kein Prozess sichtbar, Verständnis und Steuerung/Planung erschwert

Prozessmodellierung in der Praxis

- Parallelität zum Requirements Engineering
  - verschiedene Stakeholder, verschiedene Sichten, Domänenkenntnisse → Unvollständigkeit, Widersprüche, mangelnde Korrektheit
- nicht trivial
  - Komplexität, Granularität/Aufwand, projektbezogene Dynamik, Datensammlung problematisch, kulturelle Widerstände (Beschneidung der Kreativität, Ersetzbarkeit), Vermittlung des Nutzens für Prozessbeteiligte, Interessenskonflikte

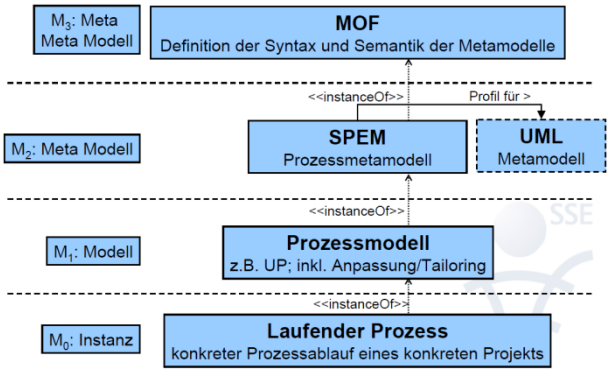
Sprachen und Werkzeuge für die Prozessmodellierung

Übersicht

- natürliche Sprache/tabellarisch
- operational: DFDs, Statecharts, UML Aktivitätsdiagramme, Petri-Netze
- deklarativ: regelbasierte Ansätze
- GP-Modellierung: EPKs
  - Aufgaben der einzelnen Rollen werden oft in Modellen vermischt, Modelle eignen sich nicht gut zur Anleitung für einzelne Rollen (in der SW-Entwicklung)
  - bei Workflow-Management-basierten Ansätzen steht das (Gesamt)Verhalten im Vordergrund (bei komplexen SW-Entwicklungs-Prozessen oft nur Beschreibung von Teilaufgaben möglich) und es werden starre Prozesse beschrieben
- wichtig: häufig Synonyme in den verschiedenen Sprachen

SPEM (Software Process Engineering Metamodell)

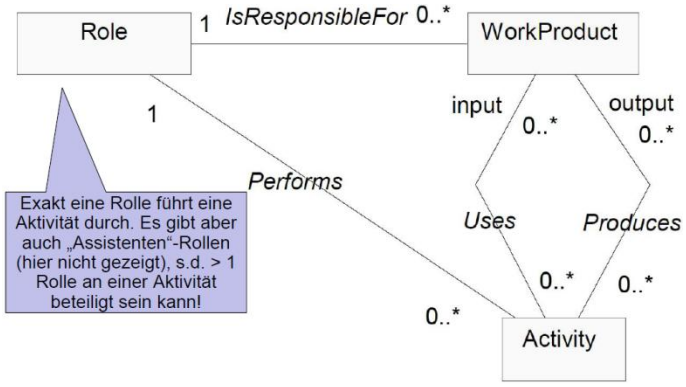
- Sprache zur Definition von Software-Prozessen
  - abstrakte Syntax und Well-formedness Rules: durch Metamodell und Profile spezifiziert
  - konkrete Syntax: Notationsmöglichkeiten werden beispielhaft beschrieben
  - Semantik: natürlichsprachlich beschrieben
- Einsatzbereich
  - Definition von Software-Prozessmodellen (insbesondre in Verbindung mit UML)
  - Instanziierung und Enactment sind nicht Bestandteil von SPEM
- in der OMG Ebenen-Architektur



alle UML-Diagramme (mit spezielleren Symbolen) sind auch für die Prozessmodellierung einsetzbar, d.h. gültige SPEM-Modelle sind auch gültige UML-Modelle

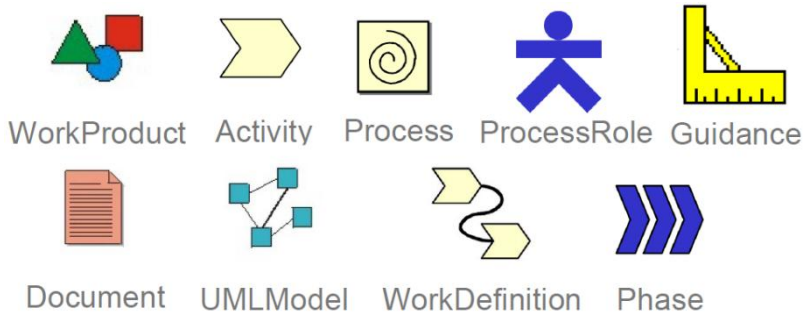
Vorteil: gesamter Sprachumfang der UML für Prozessmodellierung verfügbar (von operational bis deklarativ)

- konzeptuelles Modell
  - ein SW-Prozess in SPEM ist eine Kollaboration zwischen abstrakten aktiven Entitäten (Prozess-Rollen), welche Operationen (Activities) auf konkreten, bearbeitbaren Einheiten (Work Products) durchführen können
  - oberstes Ziel des Prozesses ist es, die Work-Products in einen wohldefinierten Zustand zu bringen



- Sprachkonzept: Guidance
  - Guidance = Ergänzung eines Prozesselements um Detailinformationen
  - Technique, UML-Profile, Checklist, ToolMentor, Guideline, Template, Estimate
- Sprachkonzept: Dependencies
  - neben existierenden UML-Beziehungen (Assoziation, Generalisierung, Komposition …) zusätzliche Beziehungen in SPEM
  - spezielle Stereotypes für UML-Assoziation: <<perform>>, <<assists>>
  - Reihenfolgeabhängigkeiten für die spezielle Usage-Beziehung <<precedes>>: Finish-Finish (FF), Finish-Start (FS), Start-Start (SS)

SPEM-Notation (SPEM 1.1)

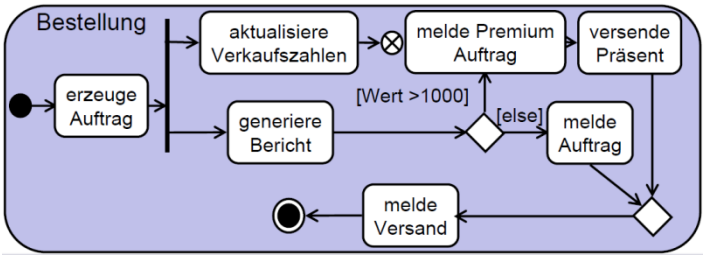


Activity entspricht Prozessschritt einer WorkDefinition

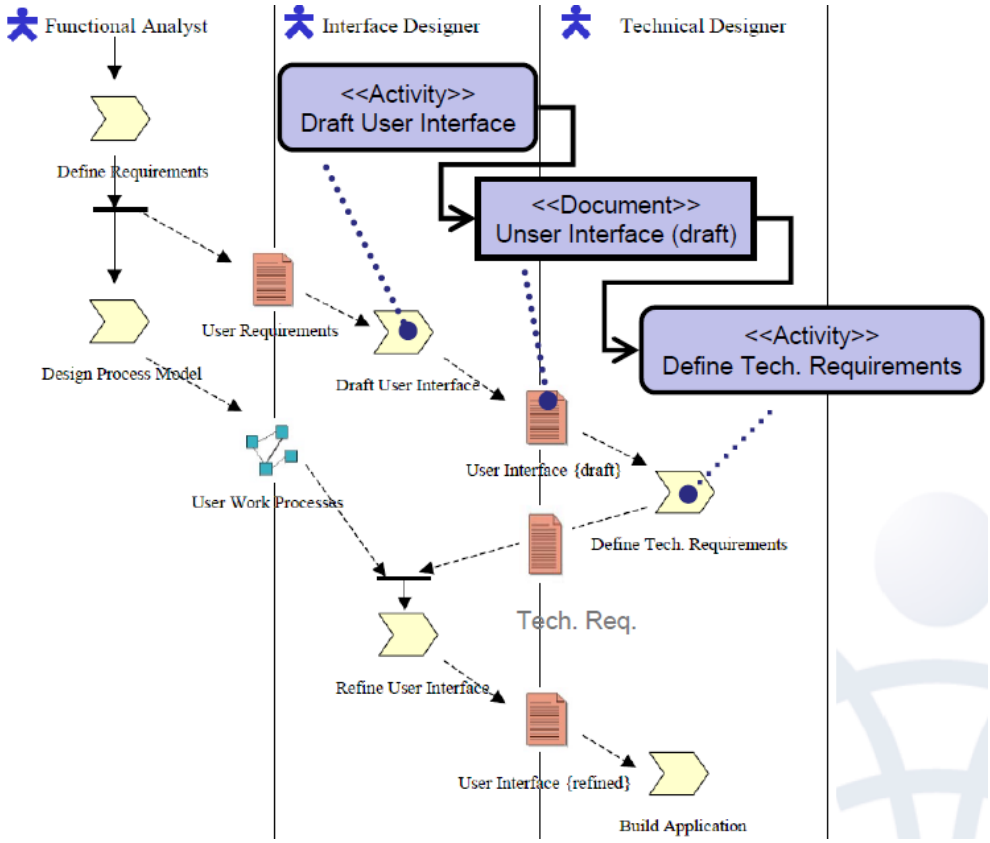
Step als weitere Verfeinerung einer Activity möglich

Anwendung von SPEM

- Exkurs: UML-Aktivitätsdiagramme (Semantik wie Petri-Netze)



- Beispiel für ein SPEM-Aktivitätsdiagramm



- Prozessmodellierung in Eclipse: Eclipse Process Framework (EPF)

Änderungen in SPEM 2.0

- bessere Trennung von Konzepten, Unterscheidung in Methoden- und Prozessbeschreibungen
- Unterstützung der Prozessanpassung (Tailoring) und optionale Prozessschritte
- initiale Unterstützung für Enactment
- Änderung der Symbole

Prozessbewertung und -verbesserung

Prinzipien der Prozessverbesserung

- Prinzipien
  - Einflüsse auf Produktqualität: Technologie, Prozessqualität, Teamqualität, Kosten/Zeit
    - Gewichtung der Faktoren abhängig von Projektcharakteristika (bei kleinen Teams sind die Menschen wichtiger, bei größeren hauptsächlich Kommunikation)
  - Prozessverbesserung impliziert nachhaltige Produktverbesserung
  - Messdaten stellen Grundlage für Verbesserung dar
  - Lokalisation der kritischen Stellen im Prozess
  - Verbesserung der kritischen Stellen durch zielgerichtete Maßnahmen
- Schritte
  1) Prozessanalyse
     - Untersuchung vorhandener Prozesse, quantitative Maßnahmen
  2) Identifikation von Verbesserungen
     - Identifizierung von Mängeln
     - Vorschlag für Verbesserungsmaßnahmen

3) Einführung der Prozessänderung
   - Einführung neuer Prozeduren, Methoden und Werkzeuge
   - Integration in vorhandenen Prozess
4) Schulung der Prozessänderung
   - Schulung der Stakeholder
5) Änderungsoptimierung
   - kleine Optimierungsmaßnahmen um absolute Effizienz zu erreichen

Evolutionary Process Improvement

Grundlegender Ansatz

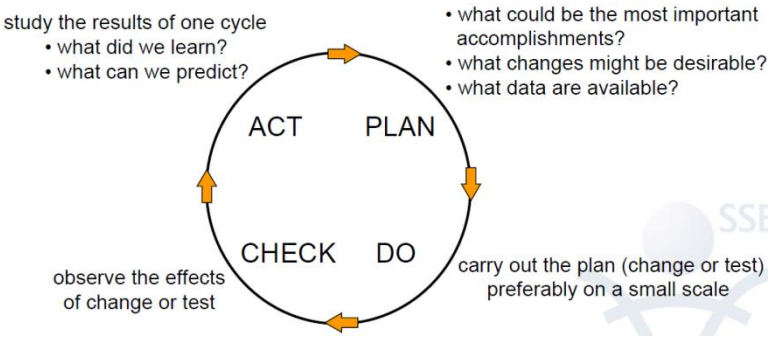
- Ziele
  - iterative Verbesserung der Prozessqualität
  - Erhöhung des Reifegrades der Prozesse
- Annahme
  - eine Organisation muss ihre Prozesse, Produkte, SW-Charakteristika und Ziele kennen, ehe sie ihren Prozess verbessern kann

  → Prozessmodellierung ist notwendig

  → Verbesserung ist kontextspezifisch
- Grundprinzip: Bottom-Up-Vorgehensweise
  - Ziele setzen unter Berücksichtigung der Charakteristika einer Organisation
  - Auswahl sinnvoller Verbesserungsmaßnahmen (differiert abhängig von Zielen und Kontext)
  - Vorgehen
    1) Einführung einer Verbesserungsmaßnahme
    2) Beobachtung der Effekte
    3) Ableitung weiterer Maßnahmen

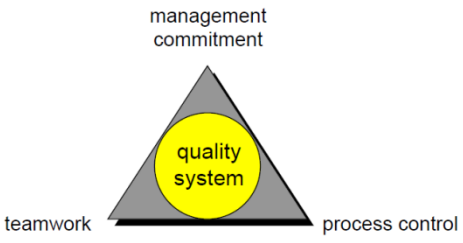
Plan-Do-Check-Act (PDCA)

- Ziel: Optimierung und Verbesserung einer einzelnen Produktlinie
- Ansatz: Feedback-Schleifen, statistische Qualitätskontrolle

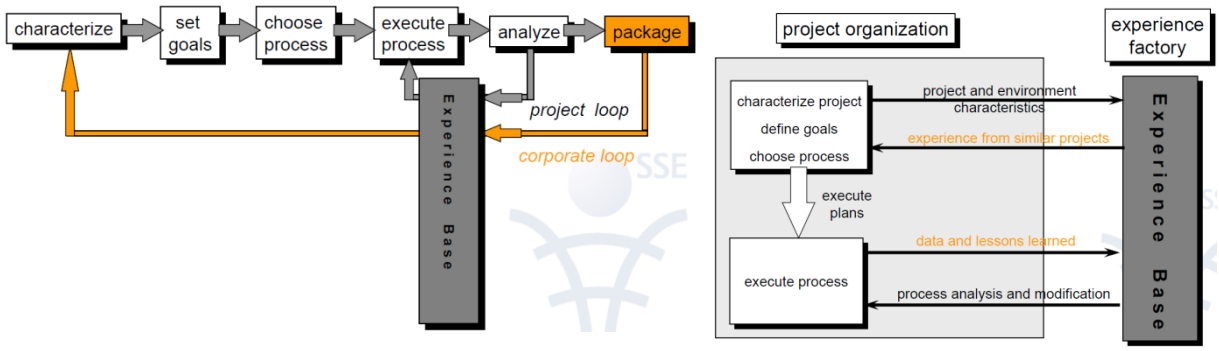


Total Quality Management (TQM)

- Erfolg eines Unternehmens wird durch Zufriedenheit des Kunden bestimmt
- Qualität ist höchstes Ziel von Mitarbeitern, über Hierarchie-Ebenen hinweg, bei der Gestaltung und Durchführung von Aufgaben



Quality Improvement Paradigm (QIP) / Experience Factory (EP)



Assessment-based Process Improvement

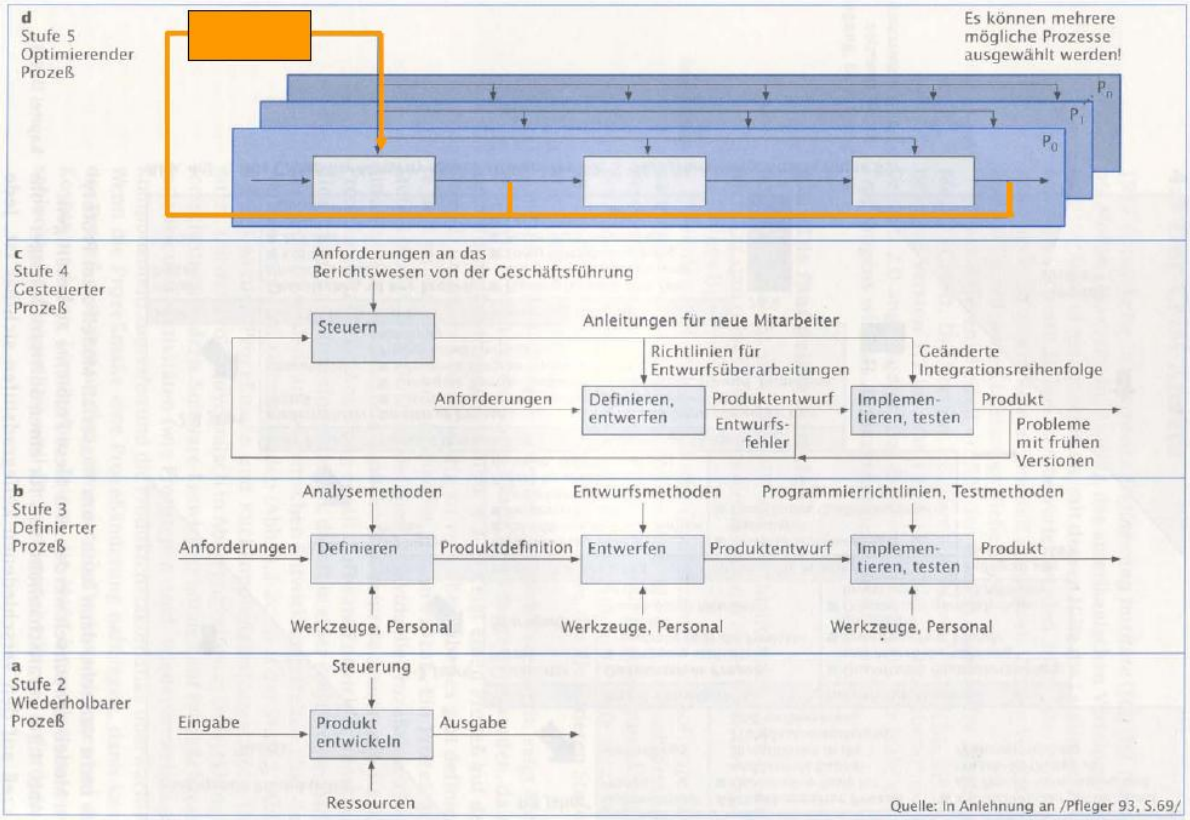
Übersicht

- Ziele
  - Bewertung des Reifegrads einer Organisation
  - Identifikation von Verbesserungsmaßnahmen (Qualität und Planbarkeit)
- Grundprinzip: Top-Down-Vorgehensweise
  - Vergleich des SW-Entwicklungsprozesses mit einem Ideal (CMMI, SPICE)
  - Prozessverbesserung ist die Eliminierung der Unterschiede (enthält Best Practices und Verbesserungspfad, jedoch keine konkrete methodische Anleitung)

CMM

- Motivation: Probleme in der Akquisition großer militärischer Systeme durch das amerik. DOD
- Ursprüngliches Ziel: Bewertung von Software-Lieferanten
- fünf Reifegradstufen (Maturity)
  - Stufe 1: Initial → unvorhersagbare Qualität, zu optimistische Schätzungen
  - Stufe 2: Wiederholbar → Termin-/Änderungskontrolle, Schätzungen auf Basis von vergangenen Projekten realistischer
  - Stufe 3: Definiert → explizite Prozesse und Methoden
  - Stufe 4: Gesteuert → messen und analysieren
  - Stufe 5: Optimierend → quantitative Planung und Kontrolle, Leistung wird kontinuierlich verbessert
- jede Reifegradstufe verbessert die Sichtbarkeit auf den SW-Prozess
  - verbessert damit auch die Steuerbarkeit und Kontrolle
- je höher der Reifegrad ist, desto
  - größer sind die erwarteten Verbesserungen bzgl. der Erreichung von Zielen
  - geringer wird der Unterschied zwischen den Plan- und den Ist-Ergebnissen
  - geringer ist die Schwankungsbreite der Ist- um die Soll-Ergebnisse herum
  - stärker sinken die Kosten, verkürzt sich die Entwicklungszeit und steigt die Qualität

- Beispiel für Prozessreifegrade



- Vorteile
  - systematische Möglichkeit zur Prozessverbesserung
  - sorgfältiges Assessment deckt Schwächen des aktuellen Prozesses auf
  - empirische Untersuchungen zeigen, dass Nutzen den Aufwand übersteigt
  - erlaubt Evaluierung des aktuellen Prozesszustandes und damit auch den Vergleich mit anderen Firmen
- Nachteile
  - kein garantierter Zusammenhang zw. hohem Reifegrad und erfolgreicher SW-Entwicklung
  - stark technikbezogen (weniger personalbezogen)
  - um eine Stufe höher zu erreichen, müssen *alle* Forderungen der niedrigeren Stufen erfüllt sein

SPICE (ISO 15504)

- europäisches Gegenstück zu CMM
- differenzierte Bewertung verschiedener Prozessbereiche und Prozesse (insgesamt 29)
- Bewertung von
  - Best Practices (Aktivitäten), Work Products (Ein-/Ausgabeartefakte), Management Practices
- differenzierte Bewertung mittels Capability Levels (Reifegradstufen)
  - Incomplete
  - Performed → Zweck des Prozesses wird erfüllt
  - Managed → Prozessausführung wird geplant und gesteuert
  - Established → Ausführung ist standardisiert
  - Predictable → Prozess ist quantitativ verstanden und kontrolliert
  - Optimizing → Prozess wird kontinuierlich verfeinert und verbessert

- Vorteile
  - Assessments dienen der Bestimmung des Reifegrads und der Identifikation von Prozessverbesserungen
  - zusätzliche Stufe 1, die besonders auch für kleine Organisationen sinnvoll ist
  - einzelne Prozesse können sich auf unterschiedliche Reifegradstufen befinden
- Nachteile
  - für Reifegradstufen 4 und 5 fehlt (bisher) die theoretische und empirische Untermauerung
  - Vergleich der Prozesse unterschiedlicher Organisationen nicht über eine einzelne Zahl möglich, sonder Vergleich über Capability-Profile notwendig

CMMI

- nach dem SW-CMM entwickelten sich die verschiedenen disziplinenspezifischen CMMs
- diese unterschieden sich in der Struktur und der Definition von Maturity/Capability
- 1998 Integration der verschiedenen CMMs zum CMMI
- CMMI ist kompatibel zu SPICE

Struktur der CMMI-Modelle

Disziplinen

- Systems Engineering (SE)
- Software Engineering (SW)
- Integrated Product and Process Development (IPPD)
- Supplier Sourcing (SS)

genereller Aufbau (model component)

- Process Area
  - Specific Goals (required)
  - Specific Practices (expected)
  - Generic Goals (required)
  - Generic Practices (required)
    - Typical Work Products, Subpractices, Notes, Discipline Amplifications, Generic Practices Elaborations, References (informative)

Process Area

- Zusammenfassung mehrerer Practices, die auf gemeinsame Goals einwirken
- z.B. Project Planning (PP), Requirements Management (REQM)
- Gruppierung in Categories, z.B. Process Management, Project Management (inklusive PP)

Goal

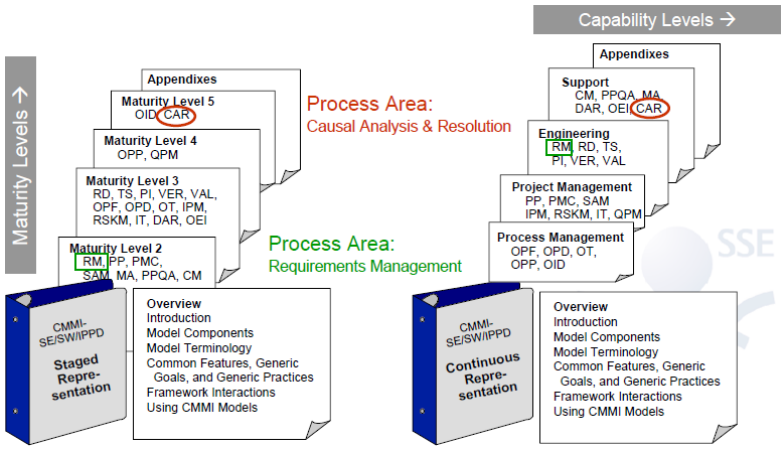
- stellt einen gewünschten Endzustand dar
- Specific Goal: gilt für eine Process Area (z.B. SG 1: Requirements are managed and inconsistencies with project plans and work products are identified.)
- Generic Goal: kann auf alle Process Areas Auswirkungen haben

Practices

- jede Practice ist genau einem Goal zugeordnet
- Specific Practice → Specific Goal, Generic Practice → Generic Goal

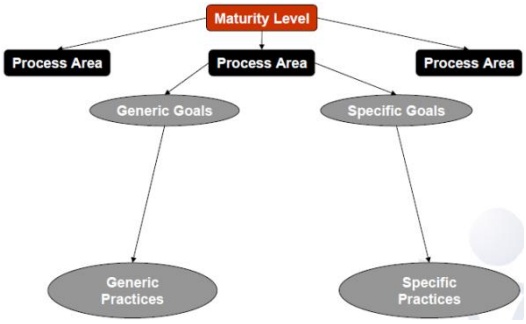
- ein Practice stellt den erwarteten, aber nicht vorgeschriebenen Weg zur Erreichung eines Goals dar
- Base Practices: alle Practices mit einem Capability Level von 1 (z.B. SP1.1-1 Develop an understanding with the stakeholders on the meaning of the requirements.)
- Advanced Practices: alle Practices mit einem Capability Level von 2 und höher (z.B. SP1.2-2 Obtain commitment to the requirements from the project participants.

CMMI Represetations



- jedes CMMI-Modell hat 2 Repräsentationen (enthalten im Wesentlichen die gleichen Infos)

Staged Representation

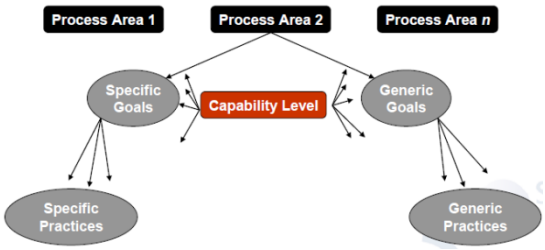


- Maturity Level zeigen, wie gut ein Unternehmen über alle Prozessbereiche hinweg performt
  - Maturity Level 1: Initial
  - Maturity Level 2: Managed
  - Maturity Level 3: Defined
  - Maturity Level 4: Quantitatively Managed
  - Maturity Level 5: Optimizing
- Vorteile
  - bietet eine Roadmap um Prozessverbesserungen zu erreichen
  - bewährter Ansatz zur Prozessverbesserung
  - Maturity Level dient als Aushängeschild
  - einfache Migration von SW-CMM zu CMMI
  - fasst die Prozessverbesserungen in eine einzelne Kennzahl zusammen

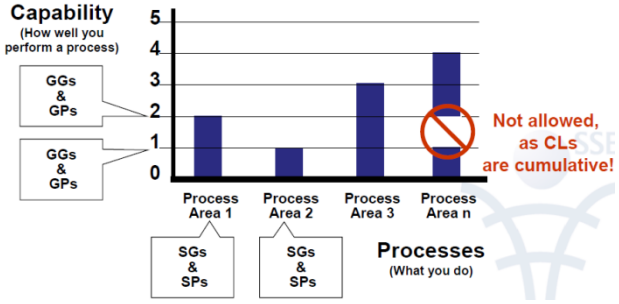
Continous Representation

- Capability Levels zeigen, wie gut ein Unternehmen in einem Prozessbereich performt
- jedes Capability Level (außer Level 0) ist einem GG und mehren GPs und SPs zugeordnet
  - Capability Level 0: Incomplete
    - Initialzustand
  - Capability Level 1: Performed → GG 1 Achieve Specific Goals
    - Base Practices sind umgesetzt
  - Capability Level 2: Managed → GG 2 Institutionalize a Managed Process
    - Projektmanagement für Prozesse (auch Verankerung in der Organisation)
    - Einbeziehung aller Beteiligten (Mgmt Commitment, Einbeziehung der Stakeholder)

    - Konfigurationsmanagement
  - Capability Level 3: Defined → GG 3 Institutionalize a Defined Process
    - Tailoring eines standardisierten Prozesses
    - Sammlung von Daten zur Prozessverbesserung
  - Capability Level 4: Quantitatively Managed → GG 4 Instit. a Quantitatively Managed Process
    - quantitative Stabilisierung von kritischen Teilprozessen (keine Kontrolle aber Vorhersagen möglich)
  - Capability Level 5: Optimizing → GG 5 Institutionalize an Optimizing Process
    - quantitative Steuerung und Verbesserung des gesamten Prozesses (Auswahl von effektiven Maßnahmen unter Kosten-/Nutzen-Aspekten)
    - kontinuierliche Erkennung und Behebung von Problemursachen
  - → Capability Level n wird erreicht, wenn GG n erreicht ist



- Capability Levels sind kumulativ und bauen aufeinander auf
- stellen eine empfohlene Reihenfolge der Verbesserungen in jedem einzelnen Prozessbereich dar
- CL-Profile ist eine Liste aller Prozessbereiche mit den entsprechenden Capability Levels



- Vorteile
  - Freiheit bei der Reihenfolge der Verbesserungen in den verschiedenen Prozessbereichen
  - stellt einen neueren Ansatz dar
  - erhöht die Sichtbarkeit von Veränderungen in den einzelnen Prozessbereichen
  - einfacher Vergleich zu SPICE

Beispiel für Generic Practice

- GP 2.7: Identify and involve the relevant stakeholders as planned.
- Subpractices
  - Identify stakeholders relevant to this process and decide what type of involvement should be practiced.
  - Share these identifications with project planners or other planners as appropriate.
  - Involve relevant stakeholders as planned.
- Elaboration für bestimmte Process Area
  - REQD: “Select relevant stakeholders from customers, end users, developers, producers, testers, suppliers, marketers, maintainers, disposal personnel, and others who may be affected by, or may affect, the product as well as the process.”

Choosing a Representation

- Wahl der Repräsentation hängt von verschiedenen Faktoren ab
  - geschäftliche Faktoren (Benchmarking), kulturelle Faktoren (Erfahrung), Altsysteme
- ein Unternehmen kann auch beide Repräsentationen verfolgen

Continous vs. Staged Representation

- allgemein gilt, um gewünschten Maturity Level zu erreichen, müssen alle Process Areas mit mindestens dem unten angegebenen Capability Level erreicht werden

Name	Abbr	ML	CL1	CL2	CL3	CL4	CL5
Requirements Management	REQM	2					
Measurement and Analysis	MA	2					
Project Monitoring and Control	PMC	2					
Project Planning	PP	2					
Process and Product Quality Assurance	PPQA	2					
Supplier Agreement Management	SAM	2					
Configuration Management	CM	2					
Decision Analysis and Resolution	DAR	3					
Product Integration	PI	3					
Requirements Development	RD	3					
Technical Solution	TS	3					
Validation	VAL	3					
Verification	VER	3					
Organizational Process Definition	OPD	3					
Organizational Process Focus	OPF	3					
Integrated Project Management	IPM	3					
Risk Management	RSKM	3					
Organizational Training	OT	3					
Organizational Process Performance	OPP	4					
Quantitative Project Management	QPM	4					
Organizational Innovation and Deployment	OID	5					
Causal Analysis and Resolution	CAR	5					

z.B. müssen für ML 4 (neben den anderen PAs) OPP und QPM auf CL 3 oder höher sein