Software Systems Engineering

Stiftung Universität Hildesheim 2003

Von Produktlinien zu Software-Ökosystemen

Prof. Dr. Klaus Schmid
schmid@sse.uni-hildesheim.de

Was sind Produktlinien?

Menge ähnlicher Systeme

= gemeinsame Features (Eigenschaften)

Was ist Produktlinienentwicklung?

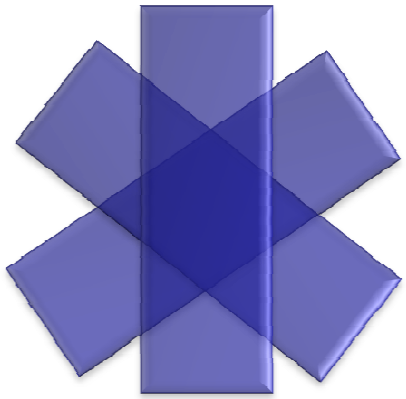
Gemeinsame Eigenschaften der Systeme
➔ *Gemeinsame Implementierung*

Clone & Own

= Wiederverwendung

Was ist die Grundidee?

- Produkt 1
- Produkt 2
- Produkt 3



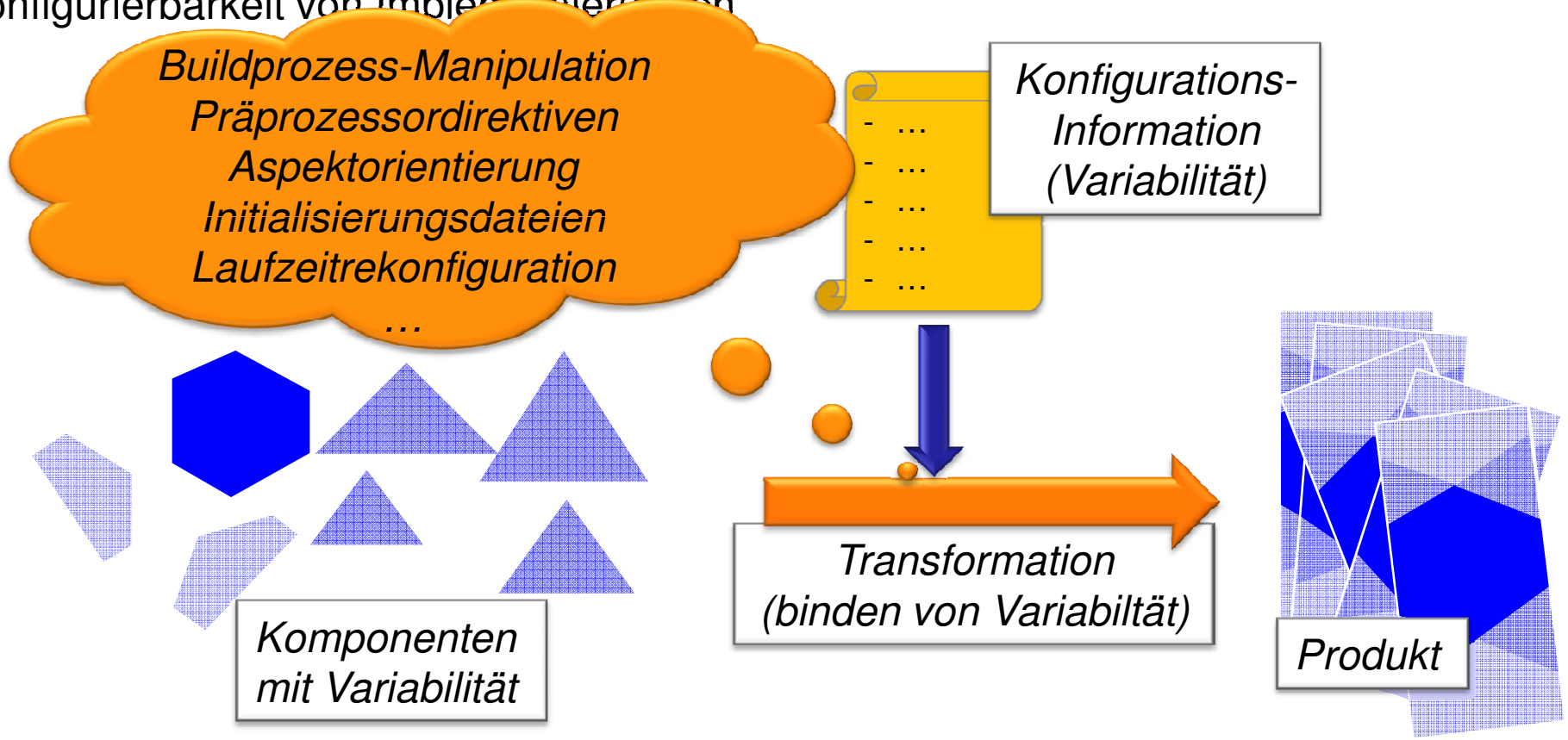
Kernidee: Ähnlichkeit von Produkten =

Gemeinsamkeit + (reg.) Variabilität + produktspezifische Teile

ein mal entwickeln | auswählbar | Einzelentwicklung

Lösungsansatz?

Konfigurierbarkeit von Implementierungen



Wird es verwendet?

AKVAsmart ASA
Argon Engineering
Asea Brown Boveri (ABB)
Axis Communications
Boeing
CelsiusTech
Cummins
Danfoss Drives
Deutsche Bank
Dialect Solutions
DNV Software
E-COM Technology
Enea
Ericsson
Eurocopter
General Motors Powertrain
Hewlett Packard
Hitachi
HomeAway
LG Industrial Systems
LSI Logic - Engenio Storage Group
Lucent Technologies
market maker Software
maxess GmbH
Mondragón Sistemas de Información (MSI)
NASA Goddard Space Flight Ctr.
NASA Jet Propulsion Laboratory
Nokia
Nokia Networks
Nortel
Overwatch Textron Systems
Philips
Philips Medical Systems
Raytheon; U. S. National Reconnaissance Office
Ricoh
Robert Bosch
Rockwell Collins
Rolls-Royce
Salion
Securitas Larm
SIEDA
Siemens
Symbian
Telvent
Testo
Thales
TomTom Automotive
Toshiba
U.S. Naval Undersea Warfare Center
U.S. Army
Wikon
…

Was bringt Produktlinienentwicklung?

- Reduktion des Code-Umfangs(>70%)
- Reduzierte Time-To-Market (2-4 fach)
- Produktivitätssteigerung (2-4 fach)
- Drastisch geringere Zahl der Fehler (defect density <50%)
- Verringerte Wartungskosten
- Geringere Zertifizierungskosten

- Verbesserte Anpassungsfähigkeit
- Schneller neue Märkte erschließen

- Vereinheitlichung der Nutzung

Software Ökosysteme vs. Produktlinien

A software ecosystem (SECO) is defined as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them.

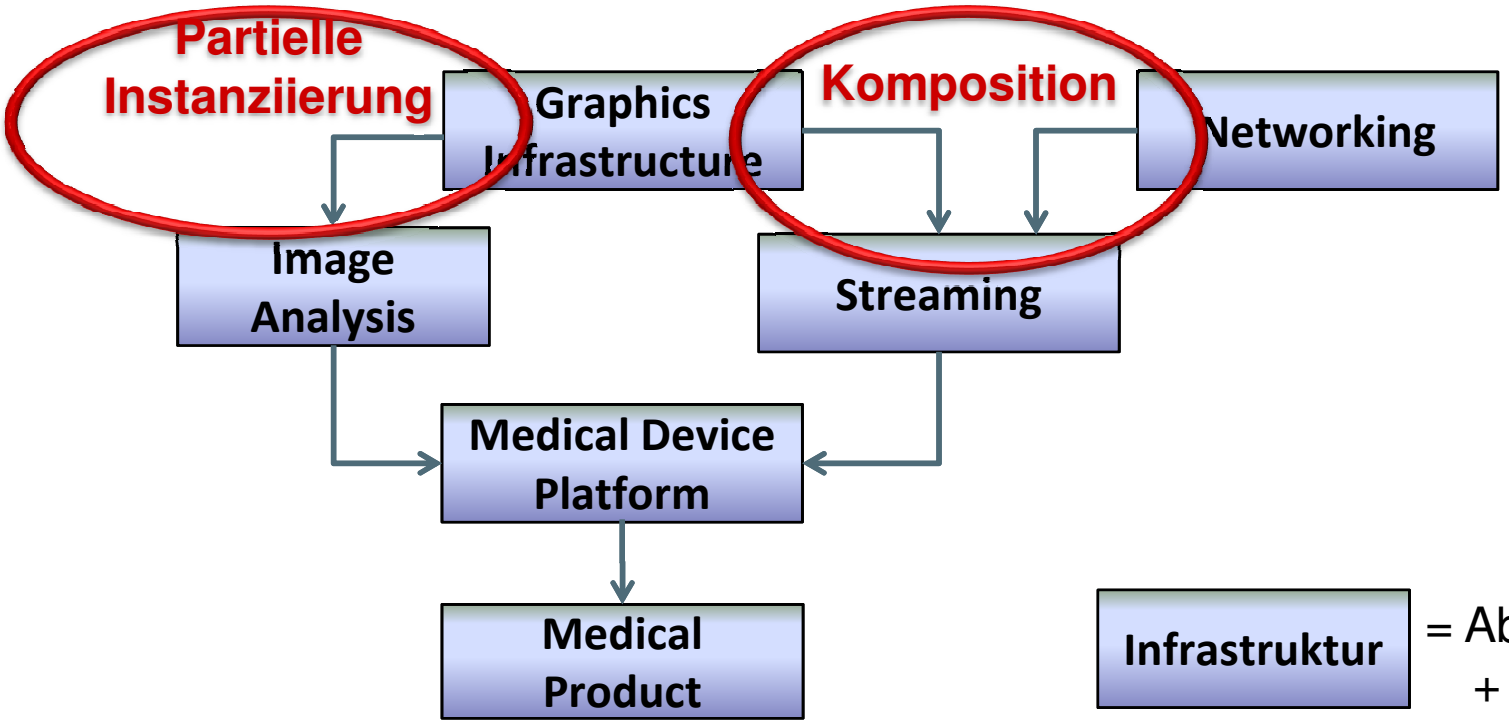
[Jansen et al. 2009]

Variantenreiche Software-Ökosysteme =

jedes Business kann selbst eine Produktlinie sein

- Abhängigkeiten zwischen Konfigurationsmöglichkeiten
- Weitergabe von Produkten als partielle Instanzen

Software-Ökosysteme = komplexe Netzwerke von Softwareleistungen



Herausforderungen:

Konfigurationsmodelle

- Konfigurationsmodelle groß /komplex
  - Verschiedenste Wertearten
  - Abhängigkeiten
- Verteilte Konfigurationsmodelle
  - Innerhalb eines Unternehmens
  - Über Unternehmen hinweg
- Partielle Konfiguration

Weitere Ziele:
- Leichte Erlernbarkeit
- Hohe Ausdrucksmächtigkeit
- Klare Trennung von Art der Transformation – nur Konfiguration

Entwicklung eines eigenen Ansatzes zur Konfigurationsmodellierung

IVML: Eine Modellierungssprache für Konfiguration – *Unterstützung von Produktlinien und Ökosystemen*

Domänenspezifische Sprache (Syntax angelehnt an Variablenbeschreibung)

- Basisdatentypen (Integer, String, ..)
- Beschränkte Typen
- Zusammengesetzte Konfigurationstypen
  - Compounds
  - Sets
  - Lists
- Konfigurationsreferenzen
- Ausdrucksmächtige Constraints
- Defaults
- Annotationen

IVML: Eine Modellierungssprache für Konfiguration – *Unterstützung von Produktlinien und Ökosystemen*

```
project contentSharing {
    enum ContentCat {text, video, audio, threeD, blob};
    typedef Bitrate Integer with (Bitrate >= 100000 and
                  Bitrate <= 400000000);

    compound ContentType {
        ContentCat cat;
        String name;
        Integer bitrate;
    }

    ContentType content;
    Bitrate     contentBitrate = 100000;
    contentBitrate <= 500000 implies content.cat==ContentCat.text;
}
```

IVML: Eine Modellierungssprache für Konfiguration – *Unterstützung von Produktlinien und Ökosystemen*

project als Basis-Einheit der Konfiguration

- Projekte haben Versionen
- Abhängigkeiten zwischen Projekten
- Schnittstellen

```
project contentSharing {
    version v0;
    import application;
    import targetPlatform with targetPlatform.version >= 1.3;
    conflicts application.version <= 1.0 with
                                    targetPlatform.version >= 2.0;
    application::name     = "myApp";
    targetPlatform::name = "myPlatform";
    application::requiredRate <= targetPlatform::providedRate;
}
```

Erfahrungen

Verschiedene „Challenge“-Workshops

- Wesentliche Erweiterung: Einführung von Referenztypen

Evaluationen in verschiedenen Projekten:

- Alle relevanten Abhängigkeiten und Konfigurationsmöglichkeiten sind darstellbar
- Führt zu einer wesentlichen Verbesserung der Dokumentation der Abhängigkeiten
- Leicht erlernbar
- Anwendungsdomänen:
  - Automotive
  - Logistikanwendungen
  - Informationssysteme

Herausforderungen:

Transformationsmodelle

- Verschiedenste Artefakttypen
  - Koordinierte Transformation
  - Unterschiedlichste Transf. Techniken
- Partielle Instantiierung
- Komposition von Produktlinien
- Verschiedenste Bindungszeiten

Weitere Ziele:
- Einfache Beschreibung
- Minimale Generierung
- Leichte Integration mit Konfiguration
- Offenheit
  (Integration existierender Techniken)

Entwicklung eines eigenen Ansatzes zur Transformations-Unterstützung

Transformationssprache: VIL (Variability Instantiation Language)

Kombiniert mehre

```
vilScript New_Product (Project source, Configuration conf, Project target) {

injectAspects(Project source, Project target, Configuration config, IvmlElement time) =
            : compileGoal() {
    sequenceOf(ClassFileArtifact) timedClassFiles =
        join(c:target.selectByType(ClassFileArtifact), exclude d:config.variables()) with
                    (c.annotationValue("Component", "decVar") == d.name() and
                     time == d.getAttribute(Test::bindingTime).value());
      map(x=timedClassFiles) {
        FileArtifact aspectFile = target_variability + "/" + (x.name()) + ".aj";
        FileArtifact template    = source_templates + "/" + time +
                                              x.annotationValue("Component", "category") +
                                                 "Template.tpl";
        createAspectFromTemplate(config, x, template, aspectFile);
      };
}
}
```

Erfahrungen

Vergleich zu Java-basierter Vorgehensweise

Projekt	Zuvor : Java	Danach : VIL
Yard management	167	94
Warehouse mgmt	469	117
SVNControl	1254	362
ScaleLog iWACS	2492	81

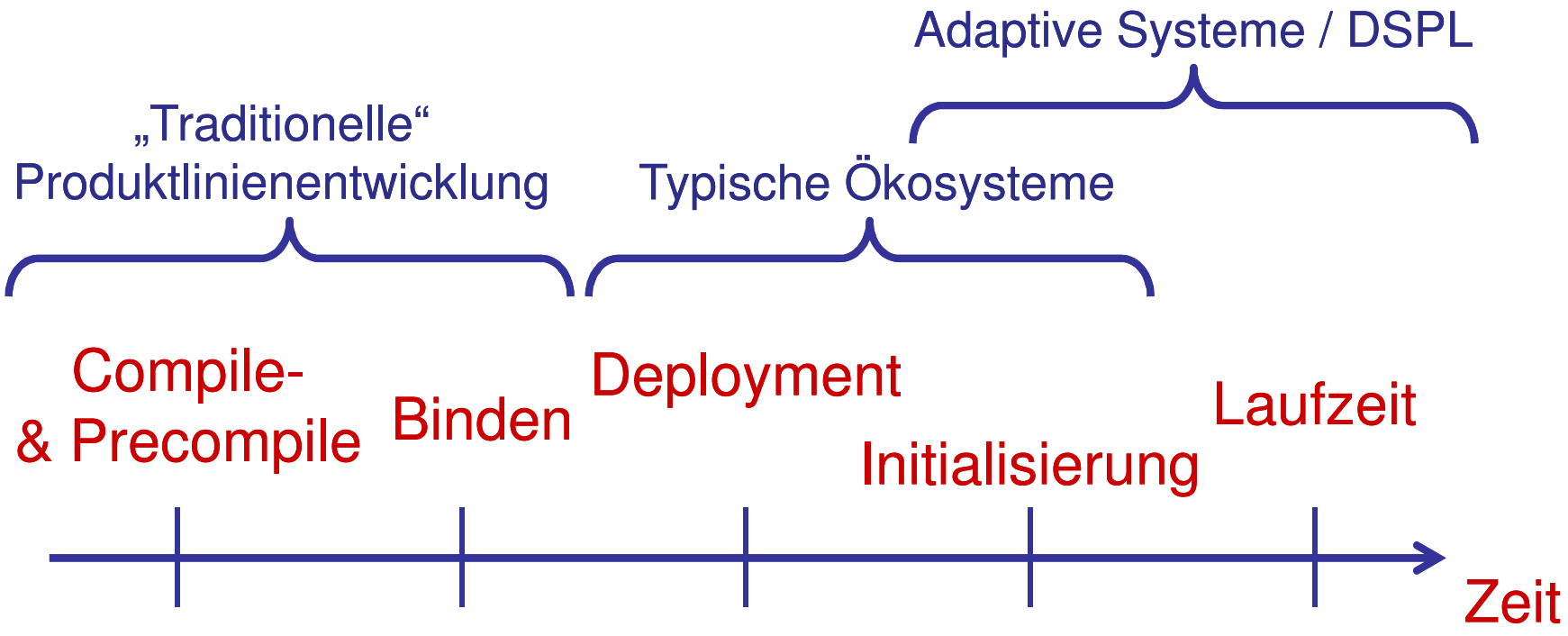
Aufbau der Werkzeuge

	DSL-basiert	Interaktiv
Konfiguration	IVML Modelling Language	Configuration View
Instantiierung	VIL & VTL Build Languages	Instantiators

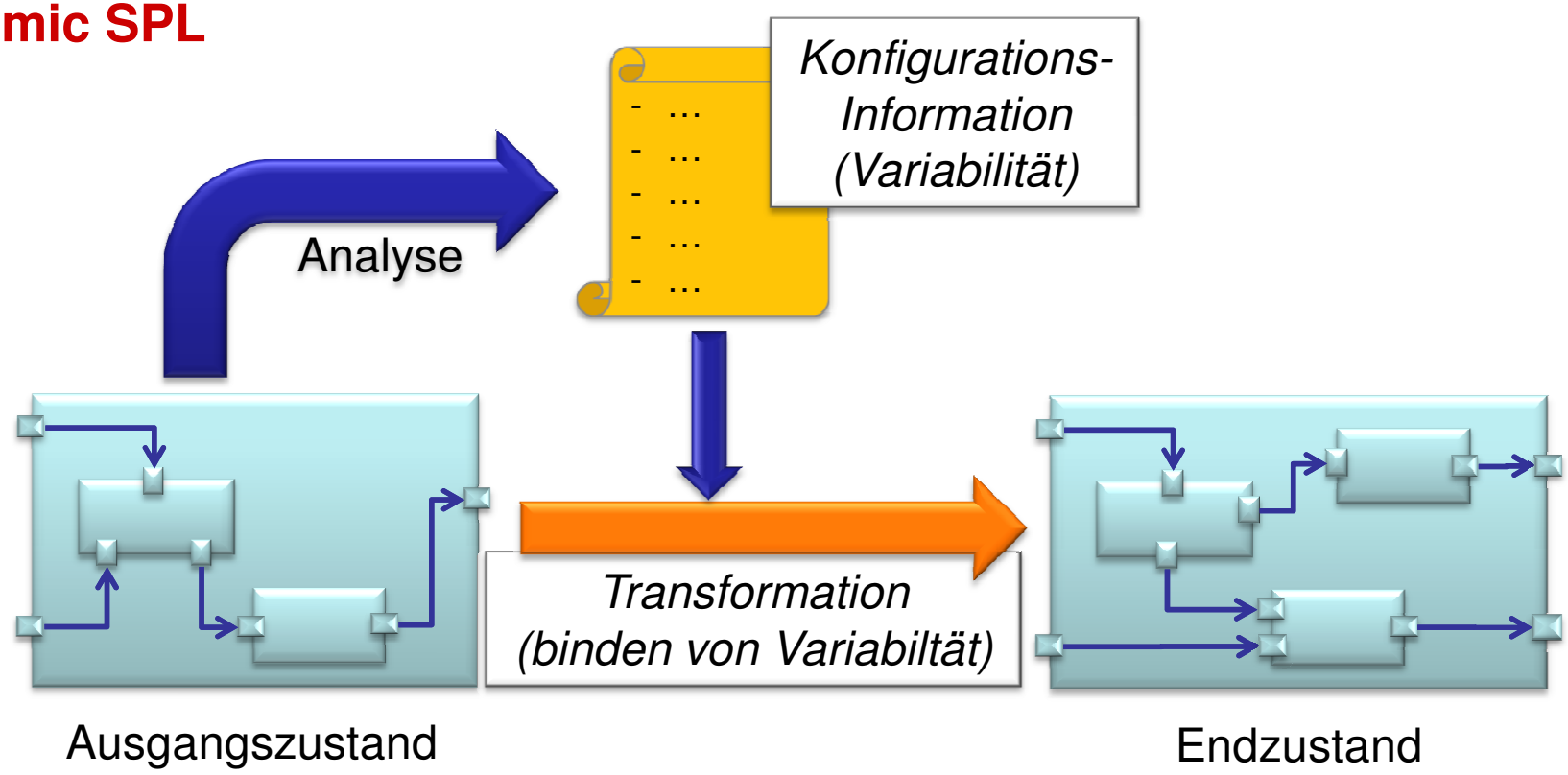
Nur Entwicklungszeit?

Bindungszeit

Ansatz deckt prinzipiell den gesamten Bereich ab



Dynamic SPL


Konfigurations-
Information
(Variabilität)
- …
- …
- …
- …
- …
Analyse
Transformation
(binden von Variabiltät)
Ausgangszustand
Endzustand

Zusammenfassung

- Einfache Beschreibung von Anpassbarkeit
- Freie Unterstützung von konfigurierbaren Transformationen
- Einfache Erlernbarkeit

- Explizite Unterstützung und Berücksichtigung von Ökosystemen (beliebige Komposition und Ableitung von Produktlinien)

- Unterstützt verschiedene Bindungszeiten
- Weiterentwicklung zu Laufzeit Adaption

EASy-Producer-Tools
sse.uni-hildesheim.de/easy-producer
ssehub.net

FRAGEN? KOMMENTARE?