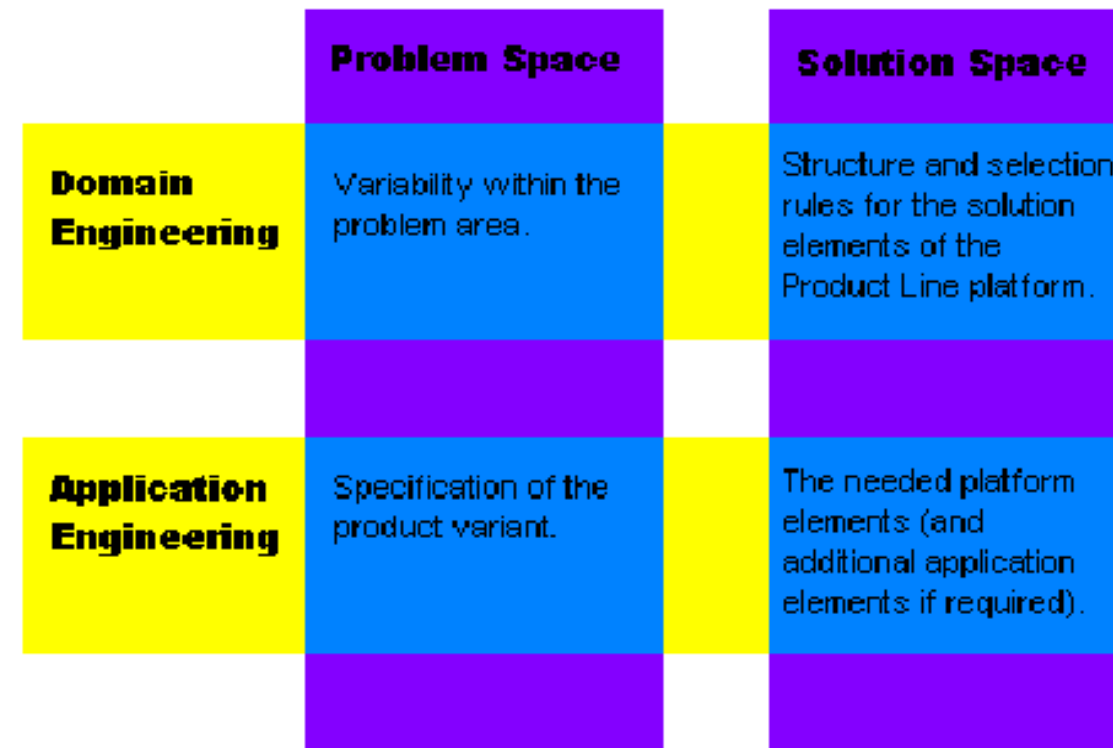


Implementierung von Software Produkt Linien

Motivation

- Behauptung: SPLs sind bereits **allgegenwärtig**
- Realisierung des Lösungs- Raums (Solution Space)
- Umsetzung von Produkt- Variabilität (Asset- Entwicklung)
- Voraussetzung: Feature- Modell (*Domain Engineering*), Lösungsarchitektur
- von Interesse: Abstraktion, Vereinheitlichung, Anpassung, Erzeugung von Code

Motivation



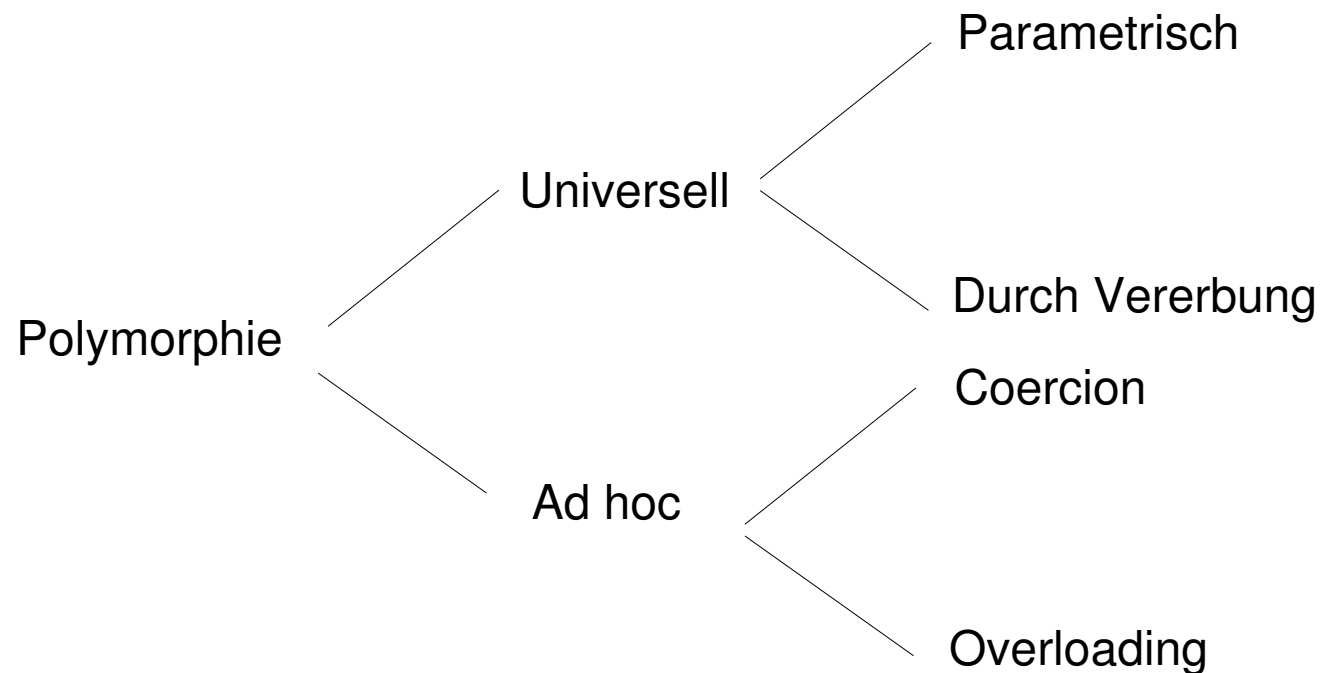
Themen

- Generische Programmierung
- Aspektorientierte Programmierung
- Generatoren
- Präprozessoren
- Verschiedenes

- Quellen

Generische Programmierung

- Benutzt Techniken von OO- Sprachen
- Polymorphie (z.B. C++, Java): verschiedene Techniken
- Parametrische Komponenten



Generische Programmierung

- SPL- Beispiel: Java »Generics« für Ausgabe- Feature

```
interface IOToolkit {
    public static void showMessage(String msg);
    // ...
}
class stdioToolkit implements IOToolkit {
    public static void showMessage(String msg) {
        System.out.println(msg);
    } // ...
}
class GUIToolkit implements IOToolkit {
    public static void showMessage(String msg) {
        // Display GUI Message box...
    } // ...
}
class statistics<T implements IOToolkit> {
    public void display() {
        T.showMessage(/* ... */);
    } // ...
}
```

Aspektorientierte Programmierung

Problem:

- Funktionen/Methoden stellen »funktionale Komponenten« dar
- Code in verschiedenen »funktionalen Komponenten« verstreut (z.B. Ressourcen- Synchronisation etc.)
- Crosscutting

Lösung:

- Zusammenfassung in »Aspekten«
- z.B. join point model

für SPLs interessant:

- Feature- Aspekt- Zusammenhang

Aspektorientierte Programmierung

- Beispiel: »tracing«- Feature mit AspectJ

```
aspect fooTracing {
    pointcut fooMethods() : execute(* foo.*(..));

    before() : fooMethods() {
        System.out.println(»enter...«);
    }
    after() : fooMethods() {
        System.out.println(»leave...«);
    }
}

class foo {
    /* ... */
}
```

Generatoren

- Automatisierung von Implementierungen insb. durch Domain Specific Languages (DSLs)
- Abstraktere Beschreibung von anpassbaren Features
- Generierter Code kann auf Feature- Auswahl/Problem angepasst sein
- Verschiedene Stufen der Implementations- Automatisierung möglich

bei SPLs:

- Vereinfachen die Anwendung einer Feature- Auswahl

Präprozessoren

- Art eines Generators
- meist: Übersetzung eines Programmtexts vor eigentlicher Verarbeitung

- Beispiel: C- Präprozessor
- Makro- abhängige Auswahl von Programmtext:
 `#if, #ifdef, #ifndef, #else, #endif`
- Makro- Definition: `#define`
- Makro- Bezeichner werden im Programmtext mit ihrem Wert ersetzt
- erlaubt Metaprogrammierung, Feature- Auswahl

Präprozessoren

- Beispiel: debug- Feature

```
#ifdef HAVE_CONFIG_H
#include »config.h« /* enthält Macros */
#endif

#ifdef USE_DEBUGGING
#define DEBUG(MSG, ...) fprintf(stderr, MSG »\n«, ##__VA_ARGS__)
#else
#define DEBUG(MSG, ...)
#endif

int main(int argc, char **argv)
{
    DEBUG(»Hello world«);
    return 0;
}
```

Verschiedenes

Nützliche Technologien:

- Autotools (autoconf, automake, libtool) zur vollständigen Umsetzung von SPLs nutzbar
- bzw. als Teil eines Build- Prozesses
- autoconf/automake Scripte beschreiben Projektarchitektur, Systemanpassungen, Systemkonfigurationen
- manche Compiler/Linker (z.B. GCC): »weak« symbols nutzbar

- SPL- Komplet- Systeme bieten verschiedene Generatoren
- Beispiel: »family models« bei pure::variants (pure::systems)

Quellen

- K. Czarnecki, U. W. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000
pure systems GmbH. Pure variants - Product Demonstration (<http://www.pure-systems.com/Walkthrough.91.0.html>)
- D. Beuche, M. Dalgamo, Software Product Line Engineering with Feature Models
- P. Simakov, Build Variety Of Consumer Products At Low Cost Using Generative Programming (<http://www.softwaresecretweapons.com/jspwiki/buildvarietyofconsumerproductsatlowcostusinggenerativeprogramming>)