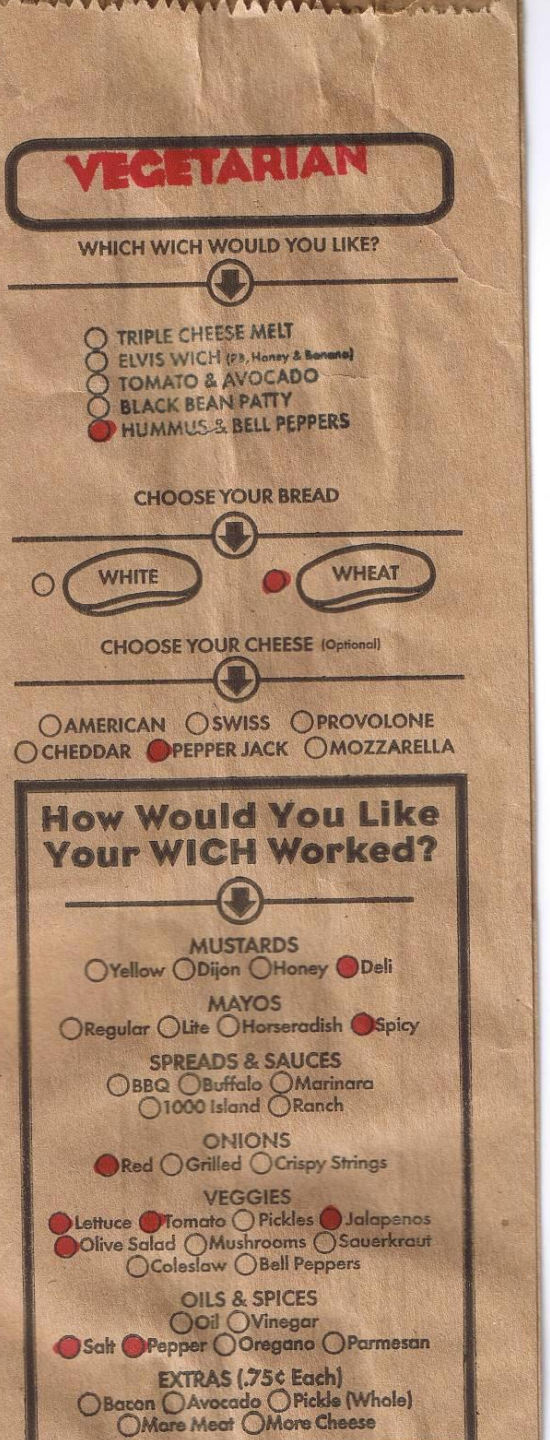
Softwareproduktlinien

Christian Kästner

Agenda

- Einfuehrung Produktlinien (inkl Herausford.)
- Wiederverwendung von Komponenten
- Domain Engineering und Automatisiertes Application Engineering
- Implementierung mit Parametern/Praeprozessoren, Crosscutting
- Implementierung mit Design Pattern und Frameworks



Feature-Oriented Product Lines

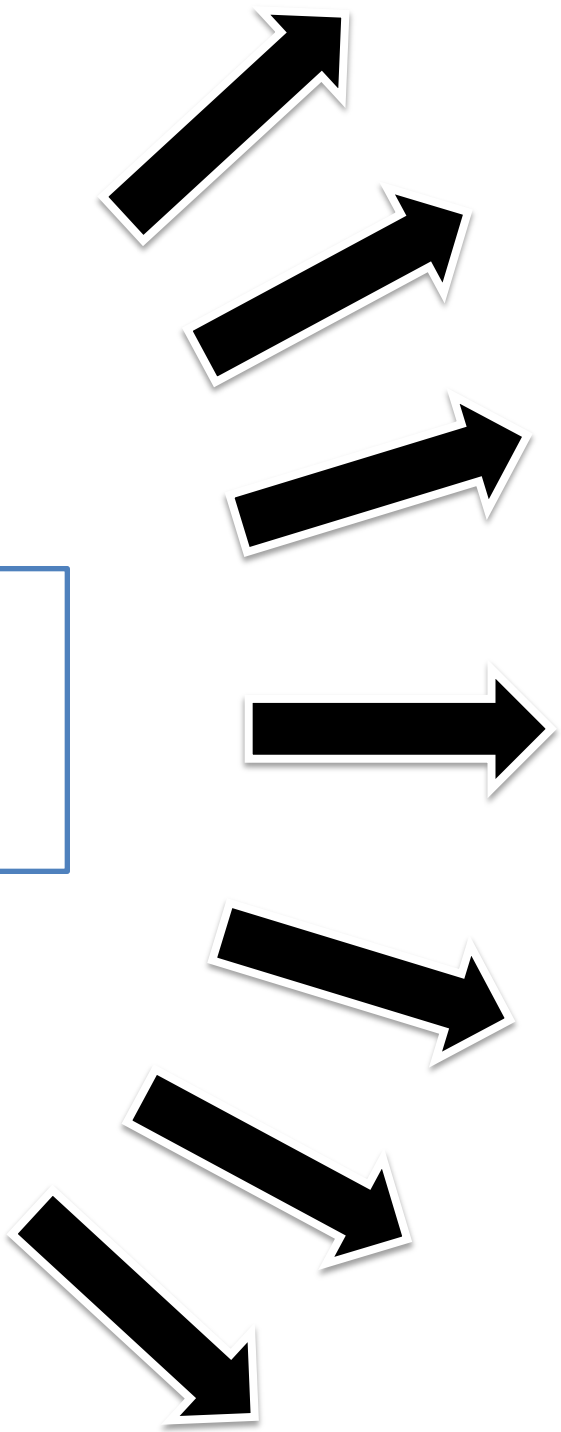
Und bei Software?

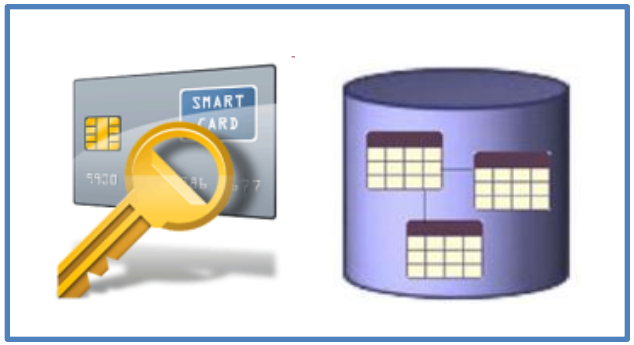
- Moderne Anwendungssoftwaresysteme sind Eier-legende Wollmilchsäue
  - Bsp.: Windows Vista, Open Office, Oracle, SAP myERP, Adobe Photoshop, Nero Burning ROM
- Spezialisierte Software und Software für eingebettete Systeme wird immer wichtiger
  - Bsp.: PDA, Handy, Sensornetze, Mikrowelle, Fernseher, Wetterstation, Auto, Chipkarten, Bordcomputer, Router, Ubiquitious Computing
  - 98% aller im Einsatz befindlichen Rechnersysteme sind eingebettete Systeme
  - Ressourcenbeschränkung und heterogene Hardware erfordert maßgeschneiderte Lösungen
  - Häufige Neuimplementierungen, lange Entwicklungszeiten, hohe Entwicklungskosten

Warum maßgeschneiderte Software?

- Ressourcenbeschränkte Systeme
  - Kosten, Energie, Platz, ….
- Individuelle Systeme versus individuelle Nutzung
  - Ungenutzte Funktionalität als Risiko
  - Wartungs- / Kontroll- / Testaufwand wächst mit Funktionsumfang
- Marketing / Preisdiskriminierung
- Schnellere Reaktion auf Marktveränderungen

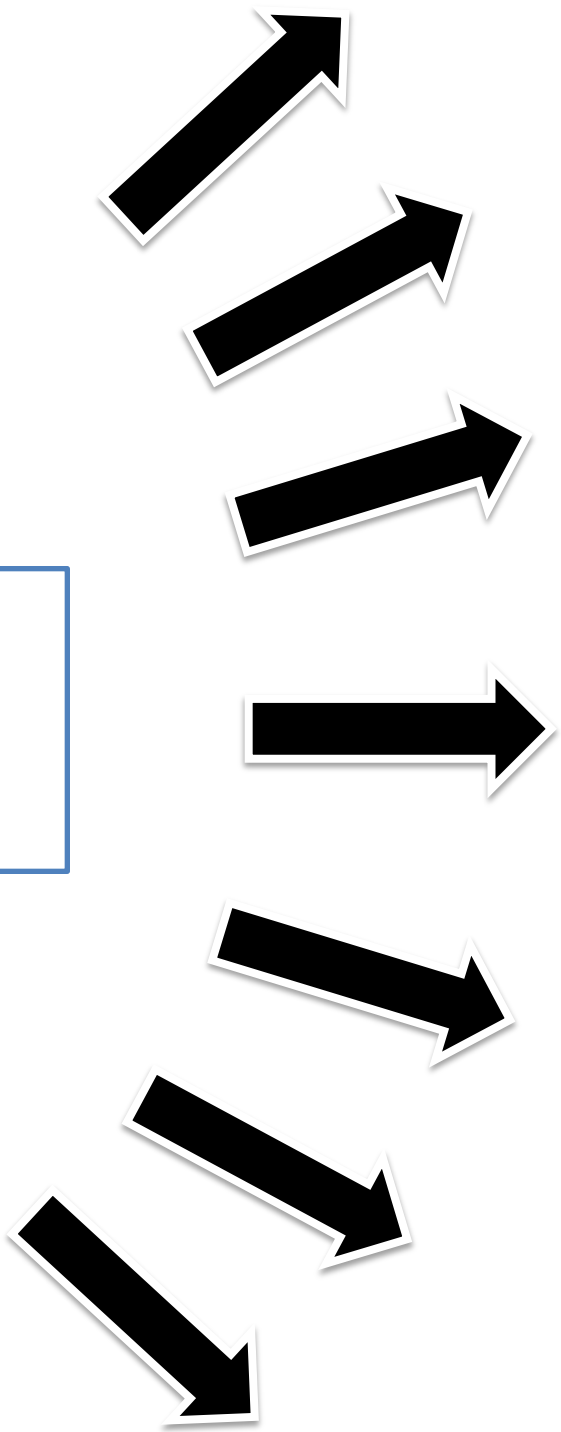
Database Engine





Printer Firmware

Linux
Kernel



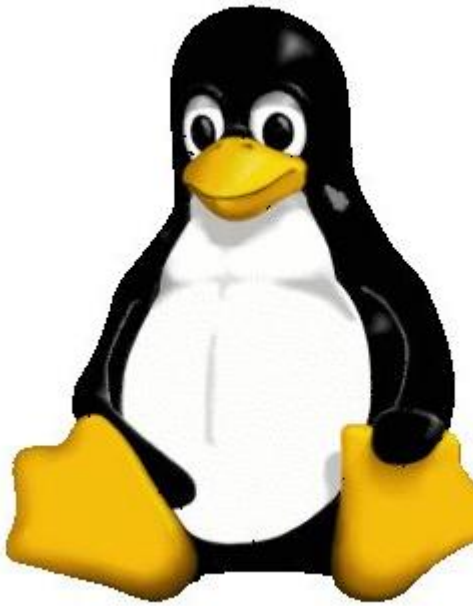
```
.config - Linux Kernel v2.6.33.3 Configuration
================================================================================
============================ Processor type and features ============================
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters
are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
excluded  <M> module  < > module capable

     [ ] Tickless System (Dynamic Ticks)
     [ ] High Resolution Timer Support
     [ ] Symmetric multi-processing support
     [ ] Support for extended (non-PC) x86 platforms
     [ ] Single-depth WCHAN output
     [ ] Paravirtualized guest support  --->
     [ ] Memtest
         Processor family (Generic-x86-64)  --->
         Preemption Model (No Forced Preemption (Server))  --->
     [ ] Reroute for broken boot IRQs (NEW)
     [ ] Machine Check / overheating reporting
     [ ] Dell laptop support
     [ ] /dev/cpu/microcode - microcode support
     [ ] /dev/cpu/*/msr - Model-specific register support
     [ ] /dev/cpu/*/cpuid - CPU information support
         Memory model (Sparse Memory)  --->
     [*] Sparse Memory virtual memmap (NEW)
     [ ] Allow for memory hot-add (NEW)
     [ ] Enable KSM for page merging
     (4096) Low address space to protect from user allocation
     [ ] Check for low memory corruption
     [ ] Reserve low 64K of RAM on AMI/Phoenix BIOSen
     -*- MTRR (Memory Type Range Register) support
     [ ]   MTRR cleanup support
     [ ] Enable seccomp to safely compute untrusted bytecode
     [ ] Enable -fstack-protector buffer overflow detection (EXPERIMENTAL)
         Timer frequency (250 HZ)  --->
     [ ] kexec system call
=====v(+)=====================================================================
================================================================================
                      <Select>    < Exit >    < Help >
```

Software Product Lines in Industry

Boeing
Bosch Group
Cummins, Inc.
Ericsson
General Dynamics
General Motors
Hewlett Packard
Lockheed Martin
Lucent
NASA
Nokia
Philips
Siemens
…

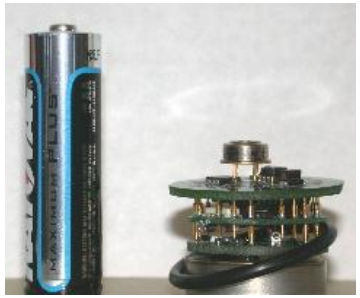
Linux-Kernel

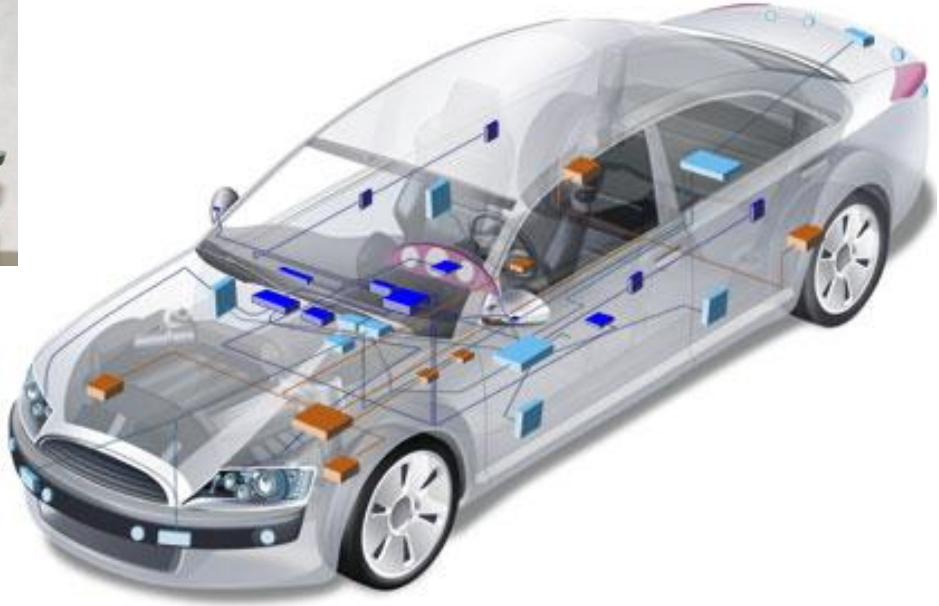
- ca. 6.000.000 Zeilen Quelltext
- Sehr weitgehend Konfigurierbar
  - > 10.000 Konfigurationsoptionen! (x86, 64bit, …)
  - Fast aller Quelltext ist „optional“



Datenbanken

- Ständig wachsendes Datenaufkommen
- Häufige Einbettung mit Ressourcenbeschränkungen





HERAUSFORDERUNGEN

Variabilität = Komplexität

33 optionale, unabhängige Features

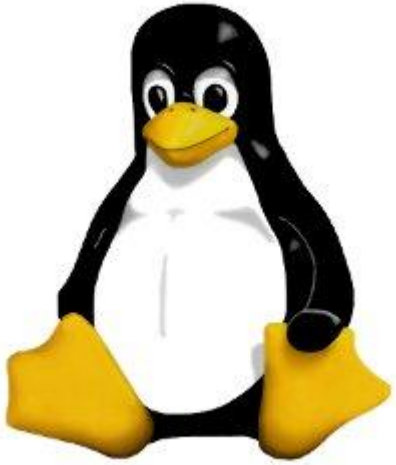
eine maßgechneiderte Variante für

jeden Menschen auf dem Planten


320 optionale, unabhängige Features
mehr Varianten als es
Atome im Universum gibt!

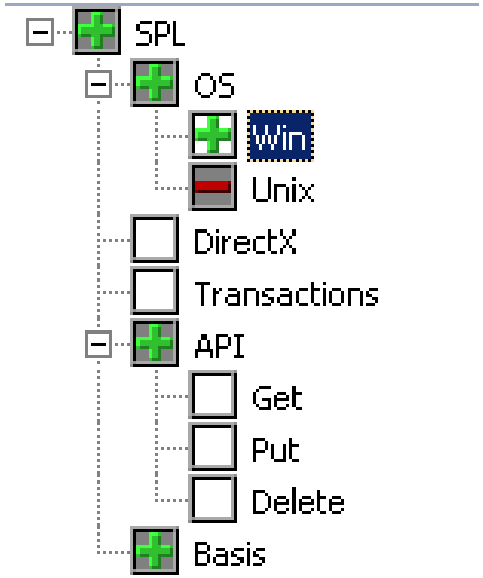
2000 Features

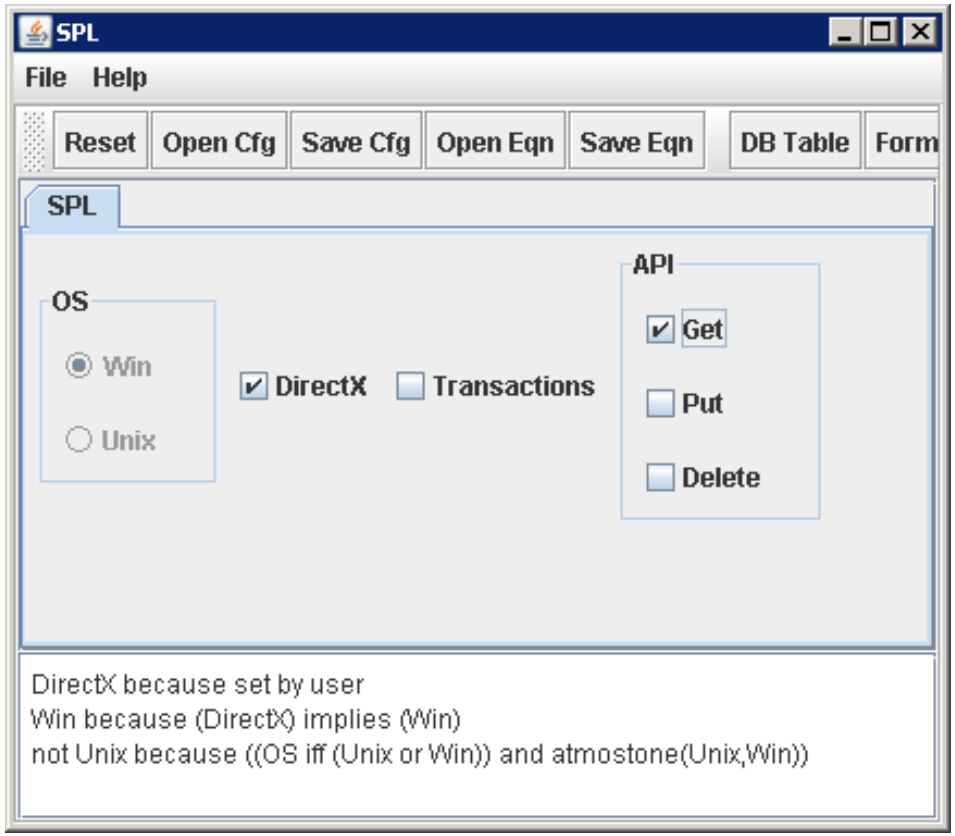
10000 Features



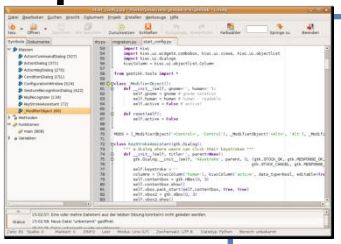
Korrektheit?

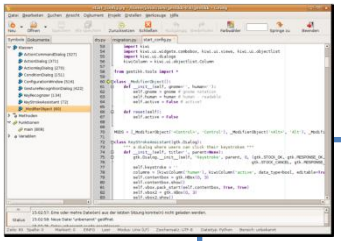
Alle Kombinationen sinnvoll?

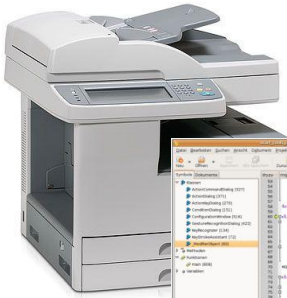


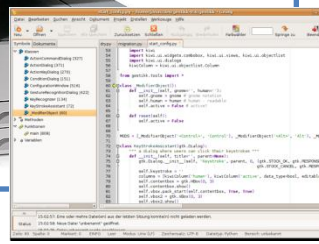


Wiederverwendung bei der Implementierung?

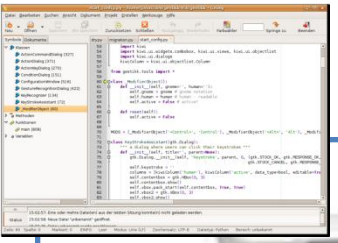


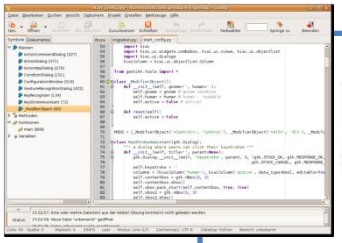






Wo Fehler korrigieren?



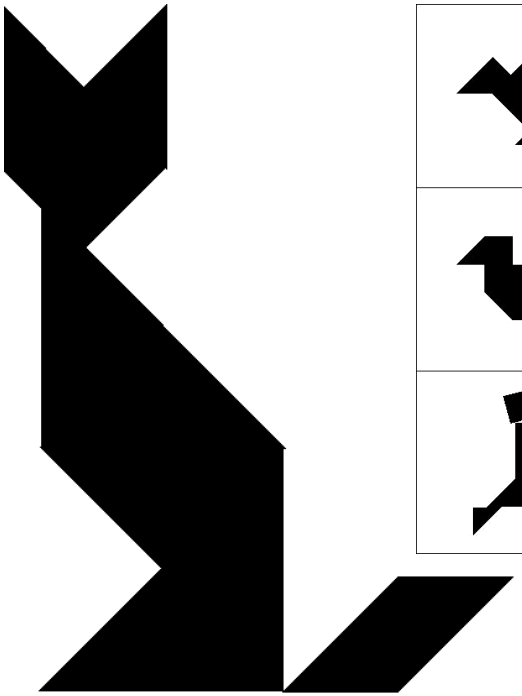


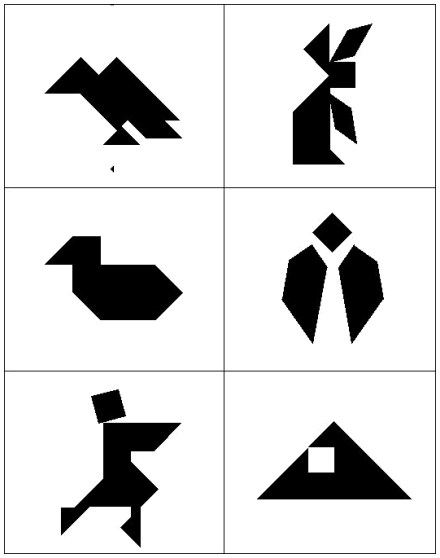
Idee: Systematische Entwicklung von Softwareproduktlinien

- Jeweils neu programmieren ist sowohl unwirtschaftlich als auch gefährlich
- Daher maßgeschneiderte Software auf Basis von Softwareproduktlinien
  - Aus wiederverwendbaren Teilen
  - Die alternative Implementierungen haben können
  - Anpassbar für spezielle Anwendungsfälle
  - Nutzbar auch unter extremer Ressourcenbeschränkung

Komponenten

- Markt für beliebige Komponenten funktioniert nicht
- Zu kleine Komponenten → hoher Aufwand
- Zu große Komp. → kaum wiederverwendbar
- Produktlinien liefern nötige Domänenanalyse
  - Welche Teilfunktionalität wird in welcher Granularität wiederverwendet
  - Systematische Wiederverwendung

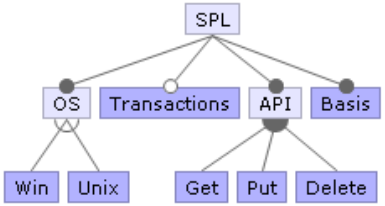




Entwurf und Implementierung von Features

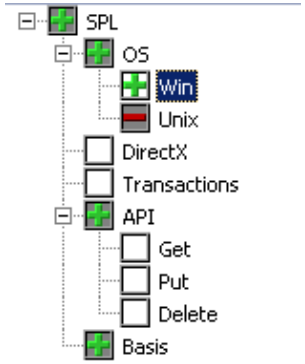
Domain Eng.

Feature-Modell



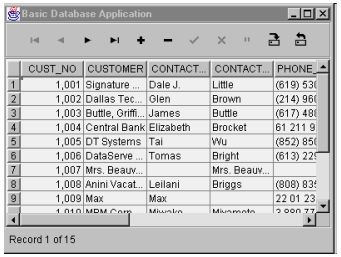
Wiederverwendbare Implementierungs-artefakte

Application Eng.



Feature-Auswahl

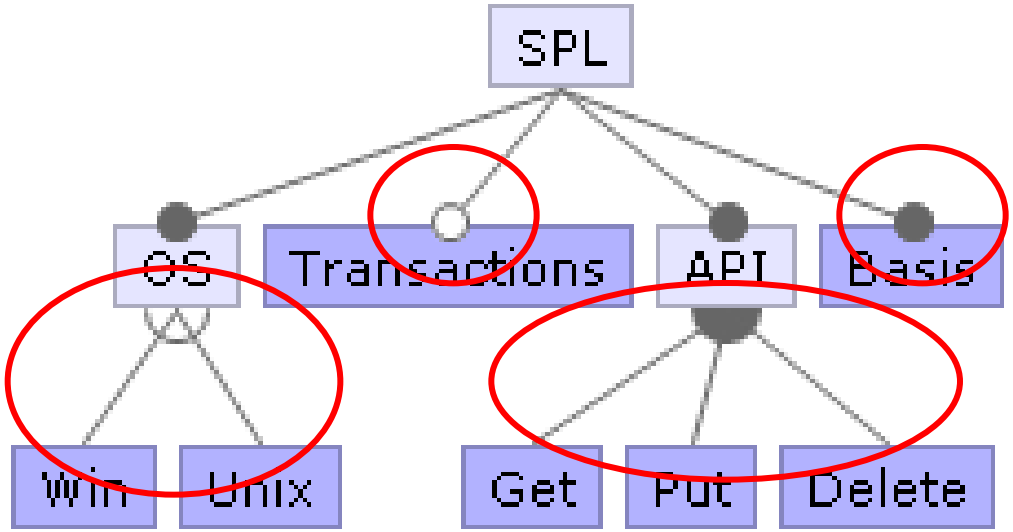
Generator



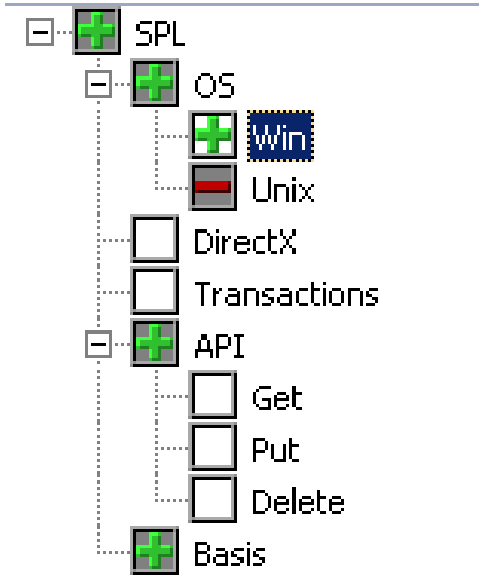
Fertiges Program

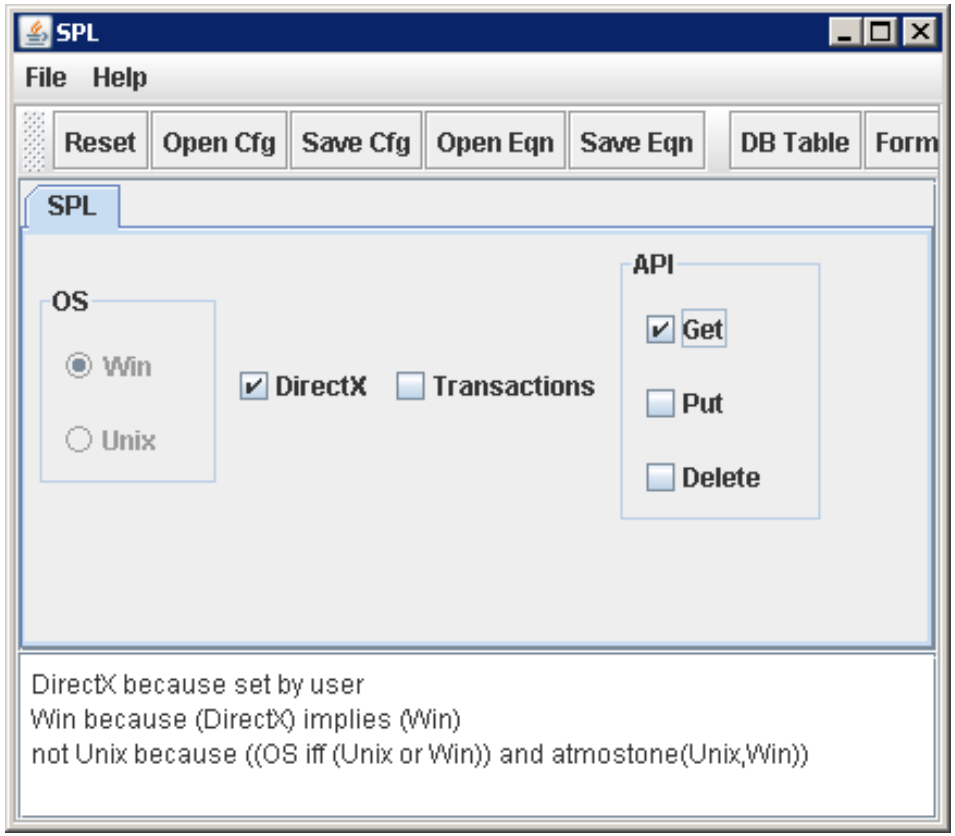
Feature-Diagramm

- Graphische Darstellung
- Hierarchische Struktur
- Kinder: optional, obligatorisch, oder, alternativ
- Features in Blättern



Konfiguration einer Variante

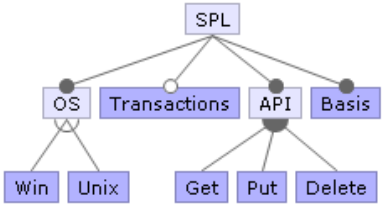




Entwurf und Implementierung von Features

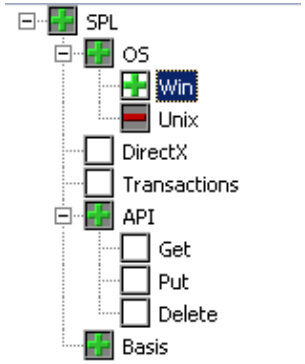
Domain Eng.

Feature-Modell



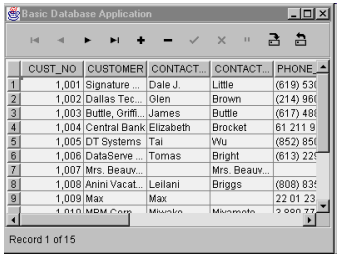
Wiederverwendbare Implementierungs-artefakte

Application Eng.



Feature-Auswahl

Generator



Fertiges Program

LAUFZEIT-PARAMETER UND BEDINGTE KOMPILIERUNG

Parameter

Administrator: C:\Windows\system32\cmd.exe

```
C:\Users\kaestner.INFORMATIK.000>dir /?
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/A[[:]attributes]] [/B] [/C] [/D] [/L] [/N]
  [/O[[:]sortorder]] [/P] [/Q] [/R] [/S] [/T[[:]timefield]] [/W] [/X] [/4]

  [drive:][path][filename]
              Specifies drive, directory, and/or files to list.

  /A          Displays files with specified attributes.
  attributes   D  Directories                R  Read-only files
               H  Hidden files               A  Files ready for archiving
               S  System files               I  Not content indexed files
               L  Reparse Points             -  Prefix meaning not
  /B          Uses bare format (no heading information or summary).
  /C          Display the thousand separator in file sizes.  This is the
              default.  Use /-C to disable display of separator.
  /D          Same as wide but files are list sorted by column.
  /L          Uses lowercase.
  /N          New long list format where filenames are on the far right.
  /O          List by files in sorted order.
  sortorder    N  By name (alphabetic)       S  By size (smallest first)
               E  By extension (alphabetic)  D  By date/time (oldest first)
               G  Group directories first    -  Prefix to reverse order
  /P          Pauses after each screenful of information.
```

Graph-Implementierungsbeispiel

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;= new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Graph-Implementierung

```
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}
```

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Diskussion

- Variabilität im gesamten Program verteilt
- Globale Variablen oder lange Parameterlisten

- Konfiguration geprüft?
- Änderungen zur Laufzeit möglich?
- Geschützt vor Aufruf deaktivierter Funktionalität?

- Kein Generator; immer alle Variabilität ausgeliefert
  - Codegröße, Speicherverbrauch, Performance
  - Ungenutzte Funktionalität als Risiko

Was fehlte?

- „If“ nicht erst zur Laufzeit auswerten
- Ganze Methoden und Klassen entfernbar machen
- Alternativen zulassen

```
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}
```

wirklich entfernen

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

entfernen

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

entfernen

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

#ifdef Beispiel aus Berkeley DB

```
static int __rep_queue_filedone(dbenv, rep, rfp)
        DB_ENV *dbenv;
        REP *rep;
        __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
        COMPQUIET(rep, NULL);
        COMPQUIET(rfp, NULL);
        return (__db_no_queue_am(dbenv));
#else
        db_pgno_t first, last;
        u_int32_t flags;
        int empty, ret, t_ret;
#ifdef DIAGNOSTIC
        DB_MSGBUF mb;
#endif
        // over 100 lines of additional code
}
#endif
```

Graph-Beispiel mit Munge

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    /*if[WEIGHT]*/
    e.weight = new Weight();
    /*end[WEIGHT]*/
    return e;
  }
  /*if[WEIGHT]*/
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  /*end[WEIGHT]*/
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
/*if[WEIGHT]*/
class Weight { void print() { ... } }
/*end[WEIGHT]*/
```

```
class Edge {
  Node a, b;
  /*if[COLOR]*/
  Color color = new Color();
  /*end[COLOR]*/
  /*if[WEIGHT]*/
  Weight weight;
  /*end[WEIGHT]*/
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    /*if[COLOR]*/
    Color.setDisplayColor(color);
    /*end[COLOR]*/
    a.print(); b.print();
    /*if[WEIGHT]*/
    weight.print();
    /*end[WEIGHT]*/
  }
}
```

```
/*if[COLOR]*/
class Color {
  static void setDisplayColor(Color c) { ... }
}
/*end[COLOR]*/
```

```
class Node {
  int id = 0;
  /*if[COLOR]*/
```

Probleme? – Verstreuter Code

Code Scattering

```
class Graph {
  Vector nv = new Vector(); Vec
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
                w Color();
  if (Conf.COLORED) Color.setDisplayColor(color);
  System.out.print(id);
 }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Probleme? – Vermischter Code

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
```

Code Tangling

```
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

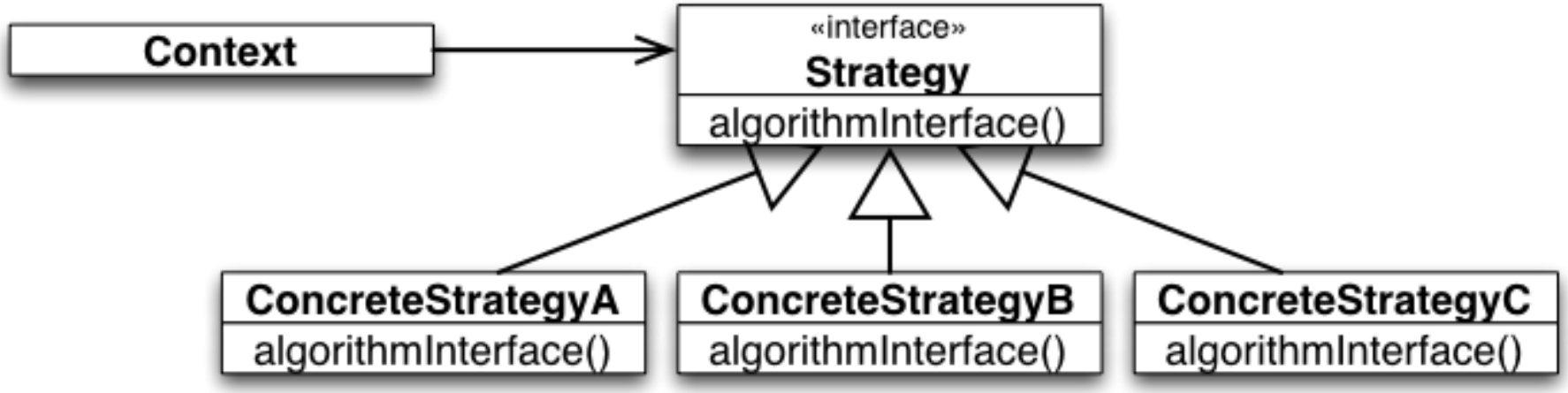
VON DESIGN PATTERN ZU FRAMEWORKS

Design Patterns

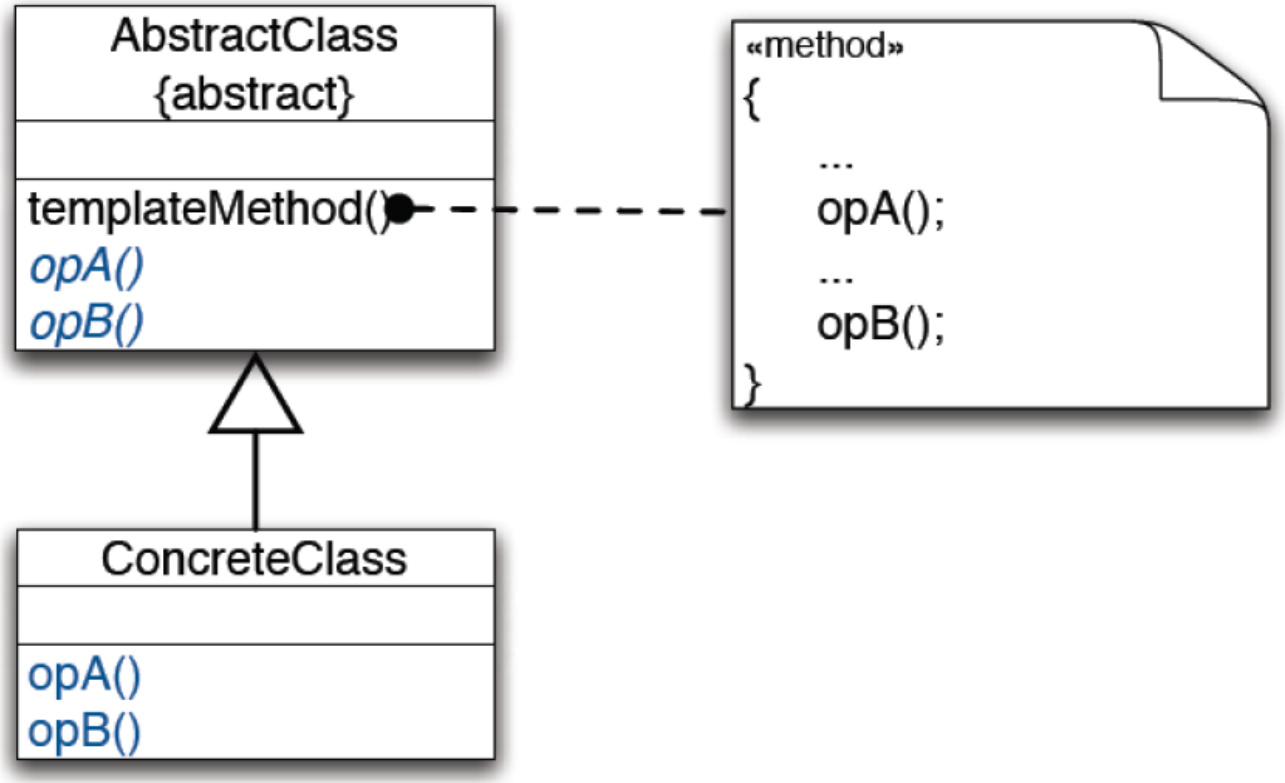
- Muster für den Entwurf von Lösungen für wiederkehrende Probleme
- Viele Design Patterns für Variabilität, Entkoppelung und Erweiterbarkeit

- Hier Auswahl:
  - Observer
  - Template Method
  - Strategy
  - Decorator

Strategy Pattern

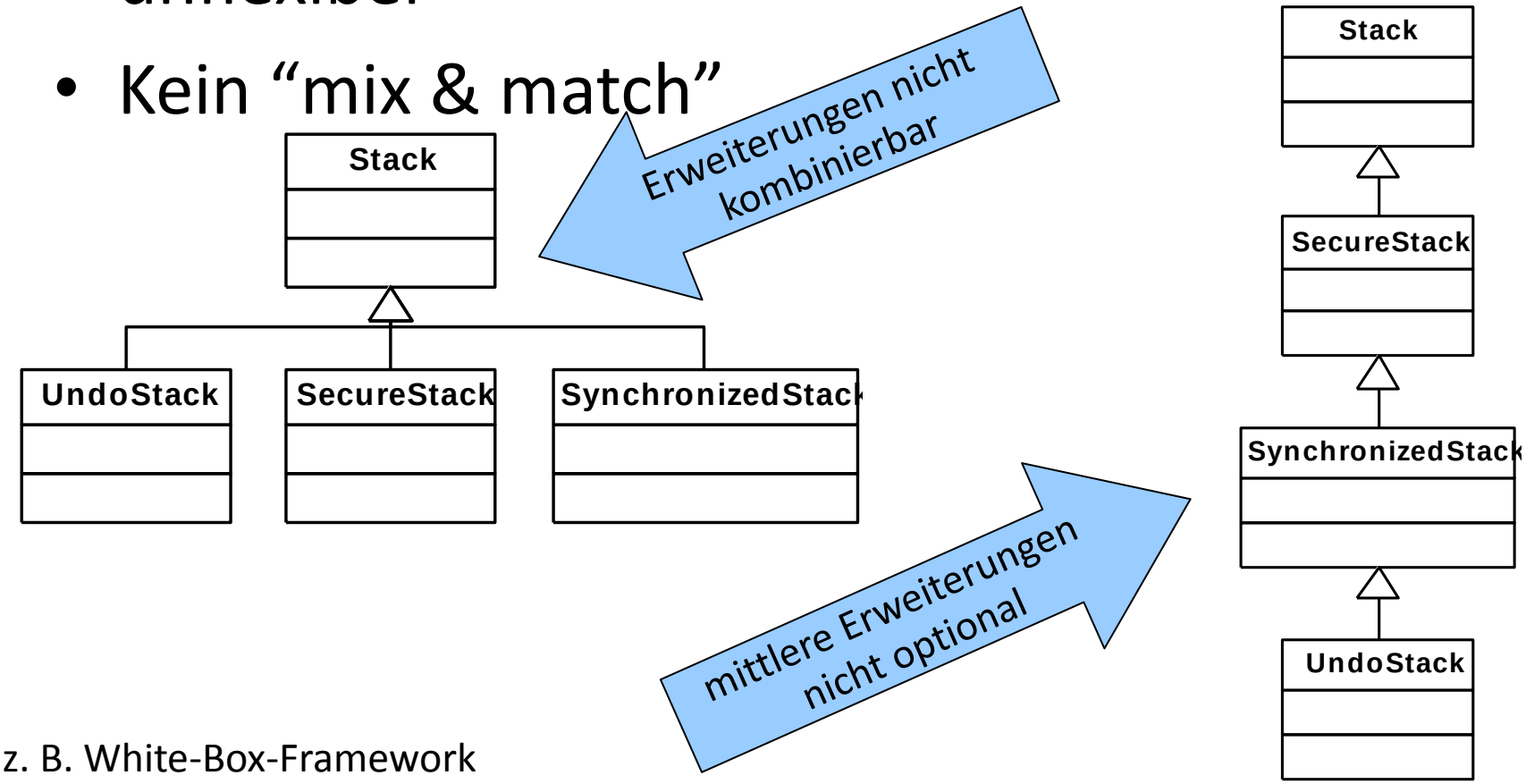


Template Method Pattern



Unflexible Erweiterungsmechanismen

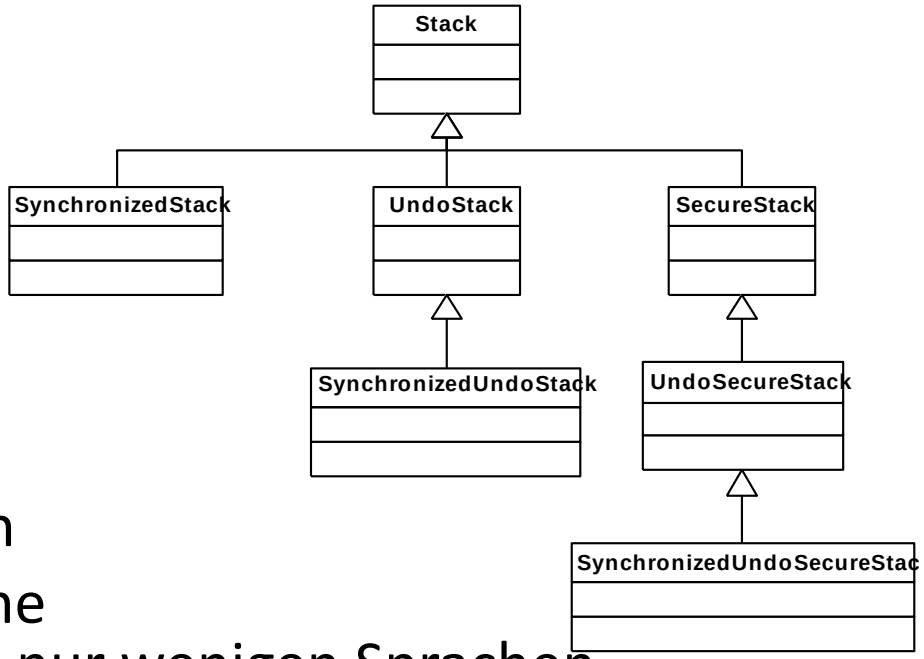
- Subklassen für Erweiterungen: modular, aber unflexibel
- Kein “mix & match”



z. B. White-Box-Framework

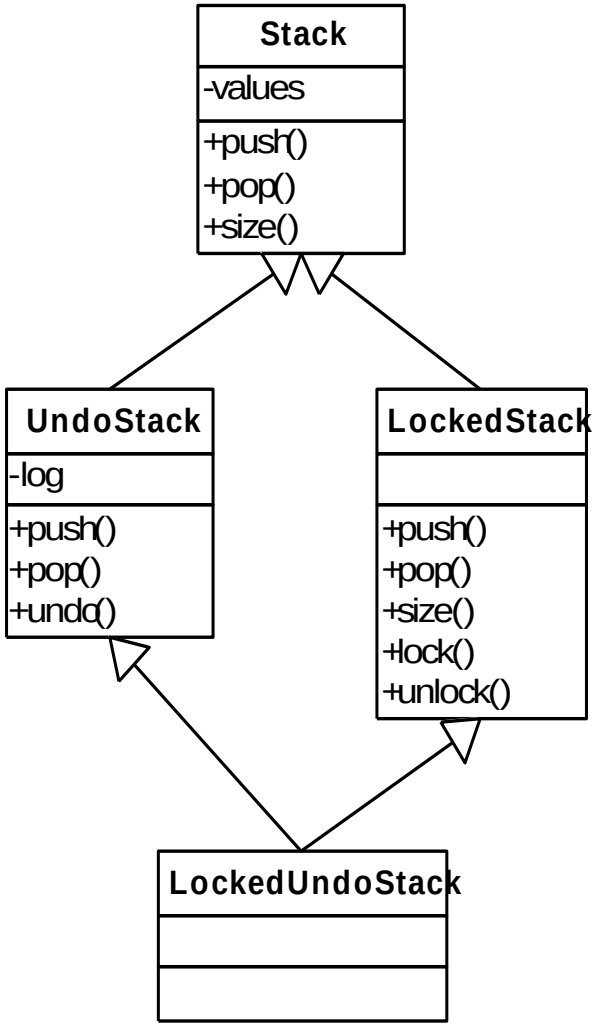
Lösung I

- Kombinierte Klassenhierarchien
  - Kombinatorische Explosion der Varianten
  - Massive Code-Replikation
  -



- Mehrfachvererbung
  - Kombinatorische Explosion
  - Aufgrund diverser Probleme (u. a. Diamant-Problem) in nur wenigen Sprachen verfügbar

Diamant-Problem

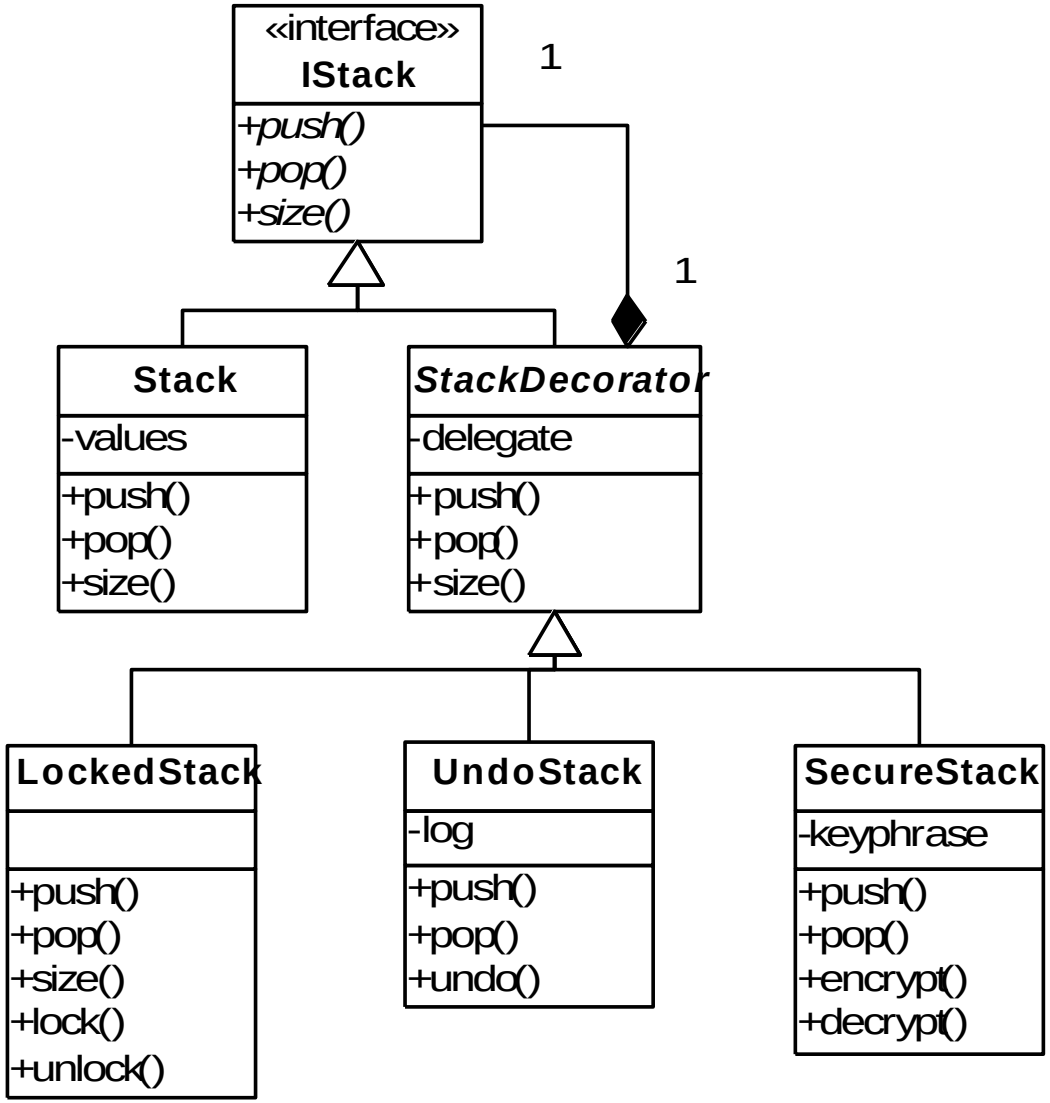


Was passiert?

```
new LockedUndoStack().pop()
```

"Multiple inheritance is good, but there is no good way to do it."
A. SYNDER

Decorator Pattern



Delegation statt Vererbung – Diskussion

- Dynamische Kombination möglich
- Erweiterungen müssen alle unabhängig sein
- Kann keine Methoden hinzufügen, nur bestehende erweitern
- Kein spätes Binden (keine virtuellen Methoden)
- Viele Indirektionen in der Ausführung (Performance)
- Mehrere Objektinstanzen bilden ein Objekt (Objektschizophrenie)

Frameworks

- Menge abstrakter und konkreter Klassen
- Abstrakte Struktur, die für einen bestimmten Zweck angepasst/erweitert werden kann
  - vgl. Template Method Pattern und Strategy Pattern
- Wiederverwendbare Lösung für eine Problemfamilie in einer Domäne
- Punkte an denen Erweiterungen vorgesehen sind: hot spots (auch variation points, extension points)
- Umkehrung der Kontrolle, das Framework bestimmt die Ausführungsreihenfolge
  - Hollywood Prinzip: „Don’t call us, we’ll call you.“

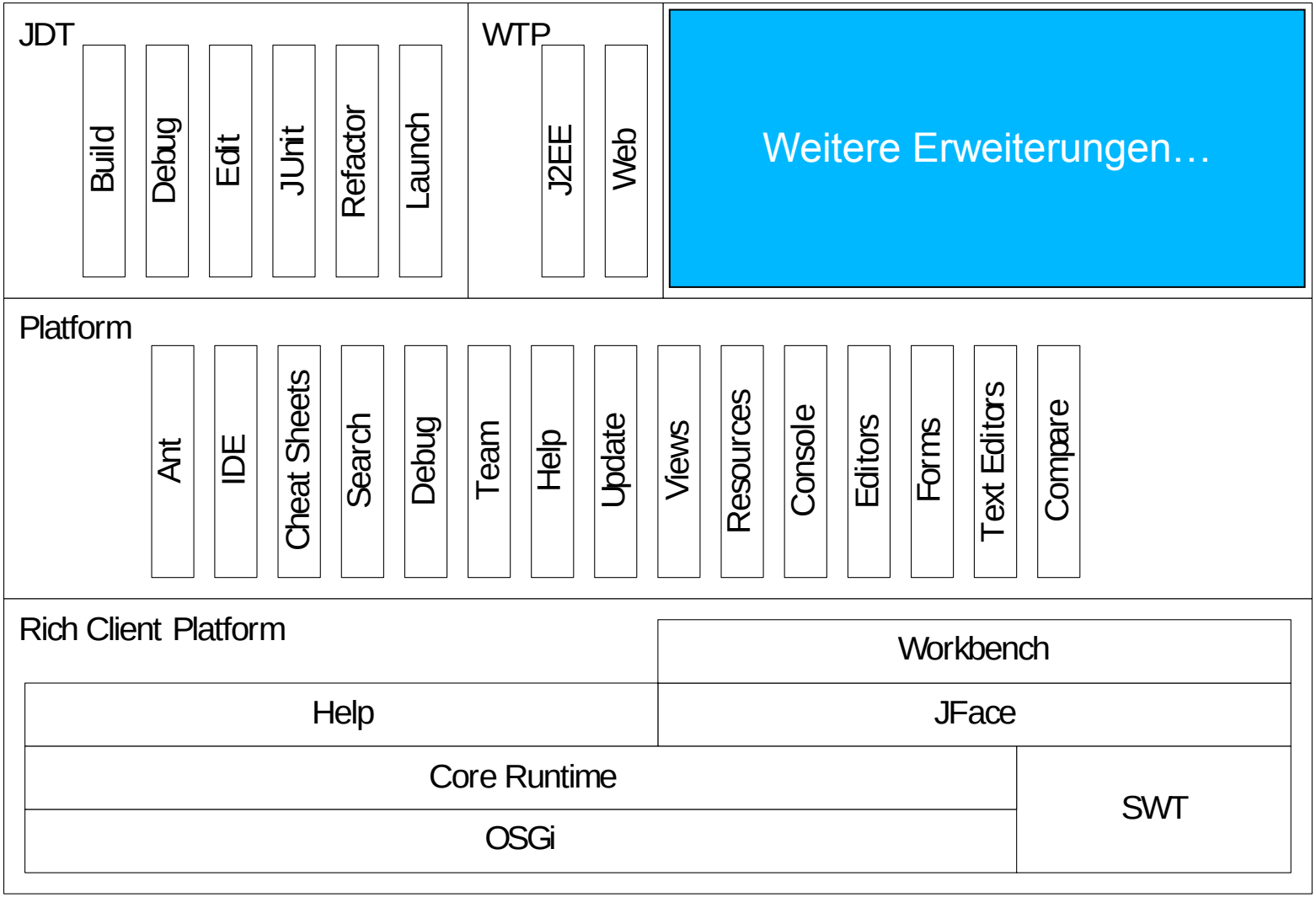
Web Portal

- Webapplikation-Frameworks wie Struts, die grundlegende Konzepte vorgeben und vorimplementieren
- Entwickler konzentrieren sich auf Anwendungslogik statt Navigation zwischen Webseiten

```
<?php
class WebPage {
        function getCSSFiles();
        function getModuleTitle();
        function hasAccess(User u);
        function printPage();
}
?>
```

```
<?php
class ConfigPage extends WebPage {
        function getCSSFiles() {…}
        function getModuleTitle() {
          return “Configuration”;
        }
        function hasAccess(User u) {
          return user.isAdmin();
        }
        function printPage() {
          print “<form><div>…”;
        }
}
?>
```

Eclipse


JDT
Build
Debug
Edit
JUnit
Refactor
Launch
WTP
J2EE
Web
Weitere Erweiterungen…
Platform
Ant
IDE
Cheat Sheets
Search
Debug
Team
Help
Update
Views
Resources
Console
Editors
Forms
Text Editors
Compare
Rich Client Platform
Workbench
Help
JFace
Core Runtime
OSGi
SWT

Framework-Implementierung: Mini-Beispiel

- Familie von Dialogen, bestehend aus Textfeld und Button

- 90% des Quelltexts sind gleich
  - Main Methode
  - Initialisierung von Fenster, Textfeld und Button
  - Layout
  - Schliessen des Fensters
  - …

Taschenrechner

```
public class Calc extends JFrame {
        private JTextField textfield;
        public static void main(String[] args) { new Calc().setVisible(true); }
        public Calc() { init(); }
        protected void init() {
          JPanel contentPane = new JPanel(new BorderLayout());
          contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
          JButton button = new JButton();
          button.setText("calculate");
          contentPane.add(button, BorderLayout.EAST);
          textfield = new JTextField("");
          textfield.setText("10 / 2 + 6");
          textfield.setPreferredSize(new Dimension(200, 20));
          contentPane.add(textfield, BorderLayout.WEST);
          button.addActionListener(/* code zum berechnen */);
          this.setContentPane(contentPane);
          this.pack();
          this.setLocation(100, 100);
          this.setTitle("My Great Calculator");
          // code zum schliessen des fensters
        }
}
```


Quelltext für alle Varianten fast gleich, nur roter Quelltext unterscheidet sich (hot sp

Black-Box Frameworks

- Einbinden des anwendungsspezifischen Verhalten durch Komponenten mit speziellem Interface (plug-in)
  - vgl. Strategy Pattern, Observer Pattern
- Nur das Interface muss verstanden werden
  - einfacher zu lernen, aber aufwendiger zu entwerfen
- Flexibilität durch bereitgestellte Hot Spots festgelegt, häufig Design Pattern
- Status nur bekannt wenn durch Interface verfügbar
- Insgesamt besser wiederverwendbar (?)

Taschenrechner

```
public class Application extends JFrame {
        private JTextField textfield;
        private Plugin plugin;
        public Application(Plugin p) { this.plugin=p; p.setApplication(this); init(); }
        protected void init() {
                JPanel contentPane = new JPanel(new BorderLayout());
                contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
                JButton button = new JButton();
                if (plugin != null)
                        button.setText(plugin.getButtonText());
                else
                        button.setText("ok");
                contentPane.add(button, BorderLayout.EAST);
                textfield = new JTextField("");
                if (plugin != null)
                        textfield.setText(plugin.getInititalText());
                textfield.setPreferredSize(new Dimension(200, 20));
                contentPane.add(textfield, BorderLayout.WEST);
                if (plugin != null)
                        button.addActionListener(/* … plugin.buttonClicked();…
            */);
                this.setContentPane(contentPane);
                …
        }
       public String getInput() { return textfield.getText();}
}
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInititalText();
    void buttonClicked() ;
    void setApplication(Application app);
}
```

Taschenrechne

Modularität?
Application kennt Plugins nicht

```
public class Application extends JFrame {
        private JTextField textfield;
        private Plugin plugin;
        public Application(Plugin p) { this.plugin=p; p.setApplication(this); init(); }
        protected void init() {
                JPanel contentPane = new JPanel(new BorderLayout());
                contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
                JButton button = new JButton();
                if (plugin != null)
                        button.setText(plugin.getButtonText());
                else
                        button.setText("ok");
                contentPane.add(button, BorderLayout.EAST);
                textfield = new JTextField("");
                if (plugin != null)

                textfie
                conter
                if (plu

        */);
                this.se
                …
        }
        public String getInput
}
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInititalText();
    void buttonClicked() ;
    void setApplication(Application app);
}
```

```
public class CalcPlugin implements Plugin {
        private Application application;
        public void setApplication(Application app) { this.application = app; }
        public String getButtonText() { return "calculate"; }
        public String getInititalText() { return "10 / 2 + 6"; }
        public void buttonClicked() {
            JOptionPane.showMessageDialog(null, "The result of "
                    + application.getInput() + " is "
                    + calculate(application.getText())); }
        public String getApplicationTitle() { return "My Great Calculator"; }
}
```

```
class CalcStarter { public static void main(String[] args) { new Application(new CalcPlugin()).setVisible(true); }
```

Weitere Entkopplung

Modularität?

Nur Plugin und InputProvider Interface

```
public class Application extends JFrame implements InputProvider {
        private JTextField textfield;
        private Plugin plugin;
        public Application(Plugin p) { this.plugin=p; p.setApplication(this); init(); }
        protected void init() {
                JPanel contentPane = new JPanel(new BorderLay
                contentPane.setBorder(new BevelBorder(BevelBor
                JButton button = new JButton();
                if (plugin != null)
                        button.setText(plugin.getButtonText())
                else
                        button.setText("ok");
                contentPane.add(button, BorderLayout.EAST);
                textfield = new JTextField("");
                if (plugin != null)

                textfie
                conter
                if (plu

        */);
                this.se
                …
        }
        public String getInput
}
```

```
public interface InputProvider {
    String getInput();
}
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInititalText();
    void buttonClicked() ;
    void setApplication(InputProvider app);
}
```

```
public class CalcPlugin implements Plugin {
        private InputProvider application;
        public void setApplication(InputProvider app) { this.application = app; }
        public String getButtonText() { return "calculate"; }
        public String getInititalText() { return "10 / 2 + 6"; }
        public void buttonClicked() {
            JOptionPane.showMessageDialog(null, "The result of "
                    + application.getInput() + " is "
                    + calculate(application.getInput())); }
        public String getApplicationTitle() { return "My Great Calculator"; }
}
```

```
class CalcStarter { public static void main(String[] args) { new Application(new CalcPlugin()).setVisible(true); }
```

Beispiel Plugin Loader (benutzt Java Reflection)

```
public class Starter {

    public static void main(String[] args) {
        if (args.length != 1)
                System.out.println("Plugin name not specified");
        else {
                String pluginName = args[0];
                try {
                        Class<?> pluginClass = Class.forName(pluginName);
                        new Application((Plugin) pluginClass.newInstance())
                                                        .setVisible(true);
                } catch (Exception e) {
                        System.out.println("Cannot load plugin " + pluginName
                                                        + ", reason: " + e);
                }
        }
    }
}
```

Frameworks für Produktlinien – Bewertung

- Vollautomatisierung möglich
- Modularität
- Praxiserprobt
- Erstellungsaufwand und Laufzeit-Overhead für Framework/Architektur
- Vorplanung nötig, Frameworkdesign erfordert Erfahrung
- Schwierige Wartung, Evolution
- Grobe Granularität oder riesige Interfaces
  - Plugin für Transaktionsverwaltung, VARCHAR oder gewichtete Kanten?

Ausblick

- Feature-Oriented Programming
- Aspect-Oriented Programming
- Feature Interaktionen
- Werkzeugunterstuetzung
- Analyse von Produktlinien

Literatur

- K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.
- R. Johnson and B. Foote, Desiging reusable classes, Journal of Object-Oriented Programming, 1(2):22-35, 1988
- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.