17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

IMPLEMENTATION 2: FRAMEWORKS, COMPONENTS

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH

READING ASSIGNMENT OCT 7

Van Ommering, Rob. "Building product populations with software components." Proceedings of the 24th international conference on Software engineering. ACM, 2002.

Bosch, Jan. "From software product lines to software ecosystems." Proceedings of the 13th International Software Product Line Conference. 2009.

LEARNING GOALS

Understand how load-time variability can be abstracted using design patterns and framework concepts; explain the role and limitations of inheritance

Choose between components and blackbox/whitebox frameworks and explain the tradeoffs

Understand the challenges of frameworks and how to mitigate them

Understand the expectations and promises of the best-of-breed approach and the practical problems

Compare frameworks and components against other implementation strategies

Explain the difference between frameworks and platforms/ecosystems

Explain modularity and relate modularity concepts against the discussed implementation strategies

MODULARITY

Feature Modularity

MODULARITY

Encapsulation (information hiding): Hiding implementation details behind an interface

Cohesion: Grouping related program constructs in one addressable unit (e.g. package, component)

Enables modular reasoning

Reduces complexity (divide and conquer)

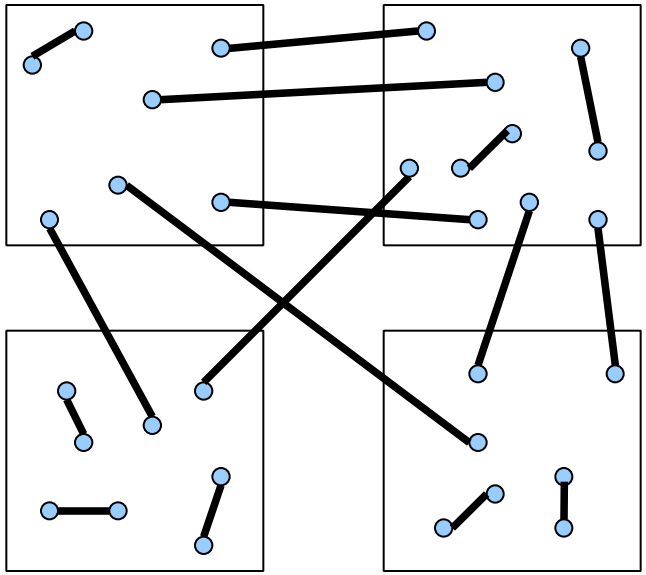
Reuse and compose pieces

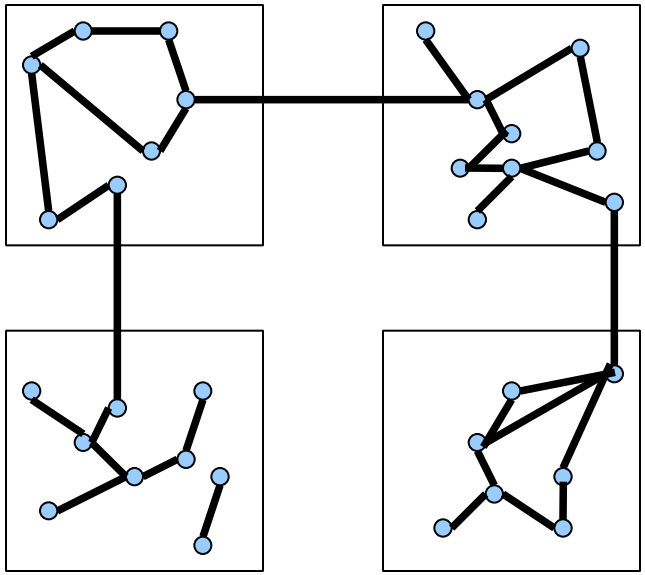
ENCAPSULATION

```
public class ArrayList<E> {
   public void add(int index, E element) {
      if (index > size || index < 0)
         throw new IndexOutOfBoundsException(
            "Index: "+index+", Size: "+size);
      ensureCapacity(size+1);
      System.arraycopy(elementData, index,
         elementData, index + 1, size - index);
      elementData[index] = element;
      size++;
   }
   public int indexOf(Object o) {
      if (o == null) {
         for (int i = 0; i < size; i++)
            if (elementData[i]==null)
            return i;
      } else {
         for (int i = 0; i < size; i++)
            if (o.equals(elementData[i]))
            return i;
      }
      return -1;
    }
   .......
}
```

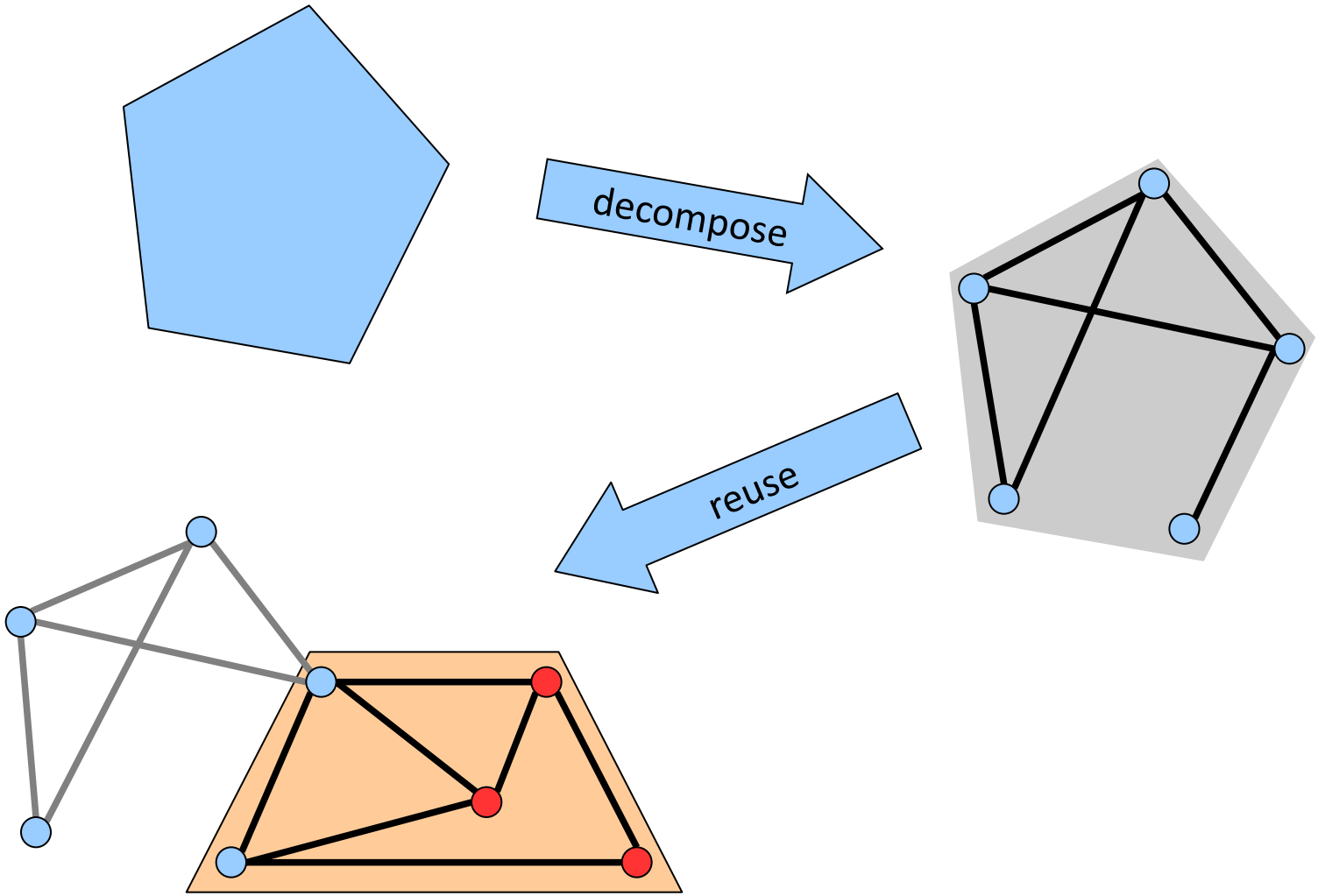
```
public interface List<E> {
   void add(int index, E element);
   int indexOf(Object o);
   ....
}
```

COHEASION AND COUPLING





DECOMPOSITION AND COMPOSITION



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() { ... } }
```

```
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() {
  }
}
```

```
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();
  }
}
```

Code Tangling

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

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

```
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;
  }
```

Code Replication

```
    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);
  }
}
```

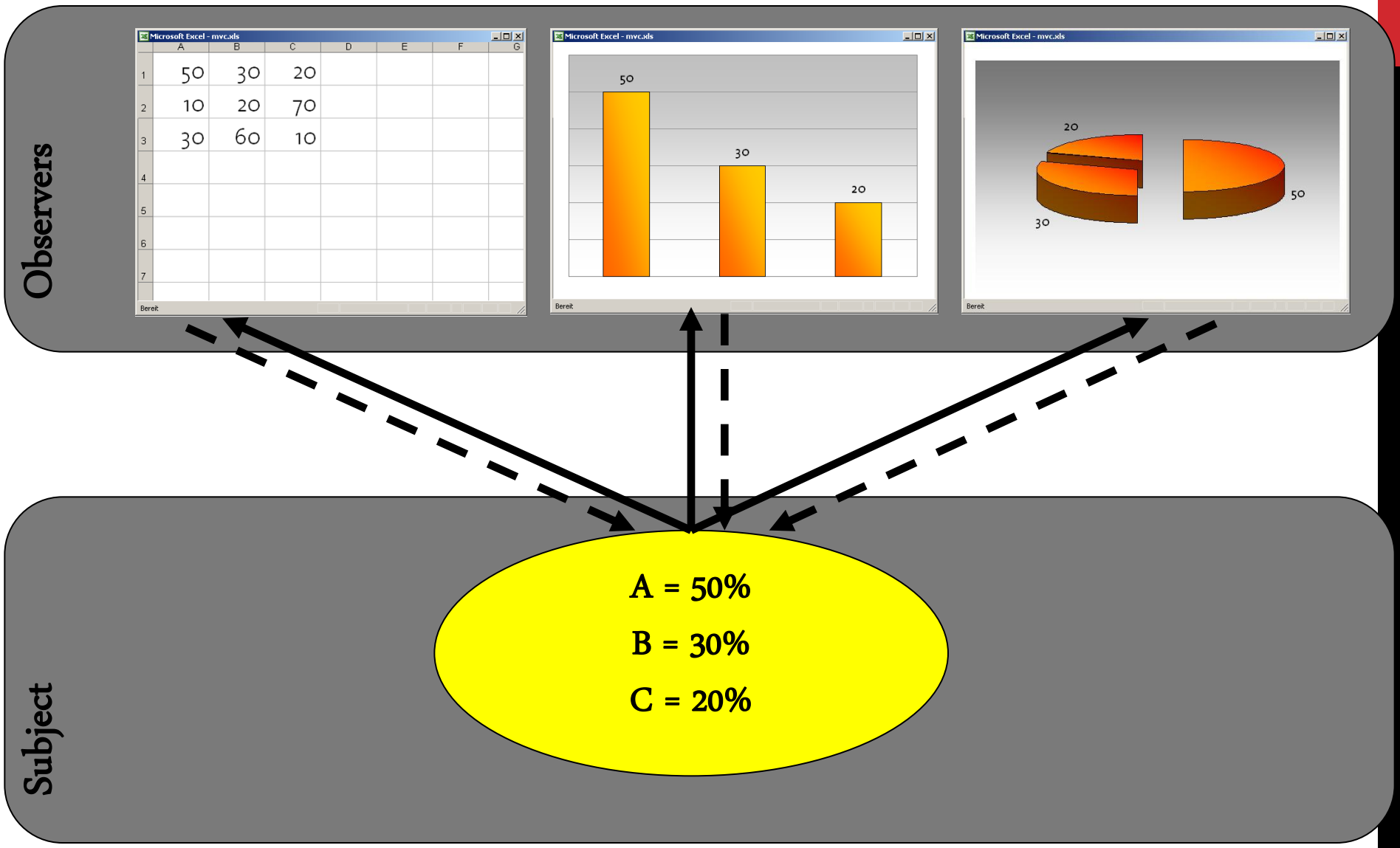
```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
   if (Conf. COLORED) Color.setDisplayColor(color);
   a.print(); b.print();
   if (!Conf.WEIGHTED)  weight.print();
  }
}
```

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

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

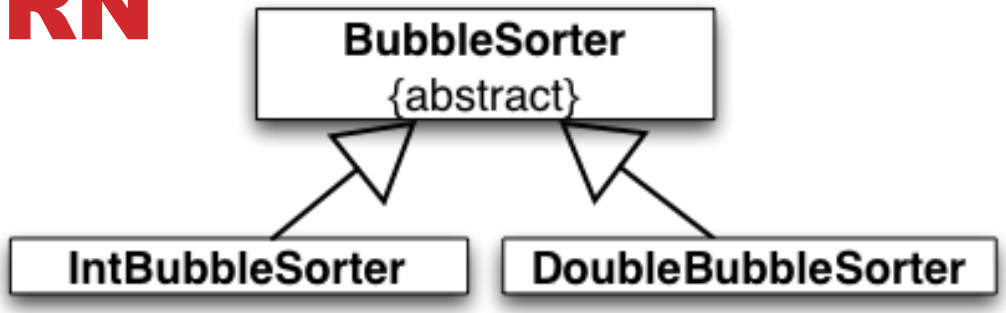
ABSTRACTING FROM RUN-TIME PARAMETERS

OBSERVER PATTERN



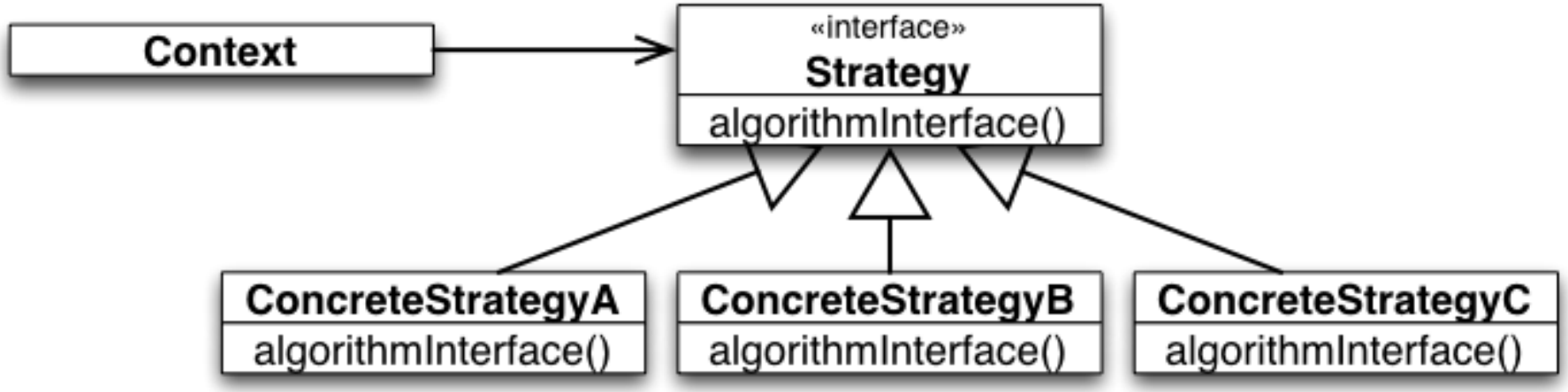
[Quelle: Meyer/Bay]

TEMPLATE METHOD PATTERN

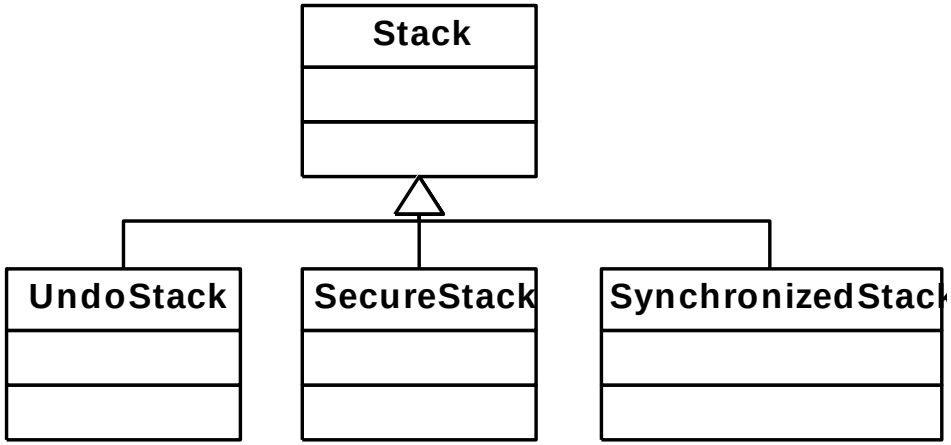


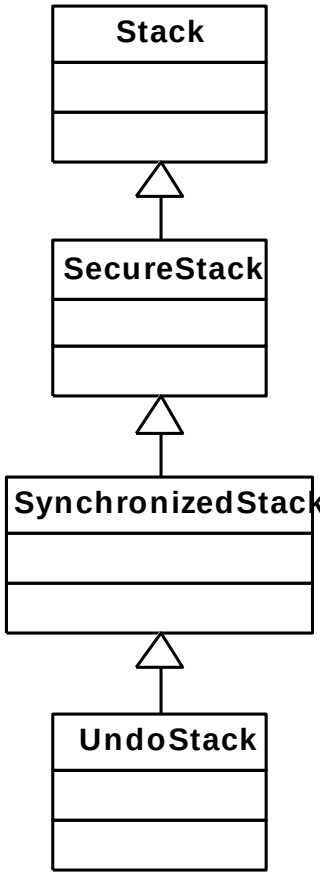

```
public abstract class BubbleSorter{
        protected int length = 0;
        protected void sort() {
                if (length <= 1) return;
                for (int nextToLast= length-2;
                        nextToLast>= 0; nextToLast--)
                        for (int index = 0;
                                index <= nextToLast; index++)
                                if (outOfOrder(index)) swap(index);
        }
        protected abstract void swap(int index);
        protected abstract boolean outOfOrder(int index);
}
```

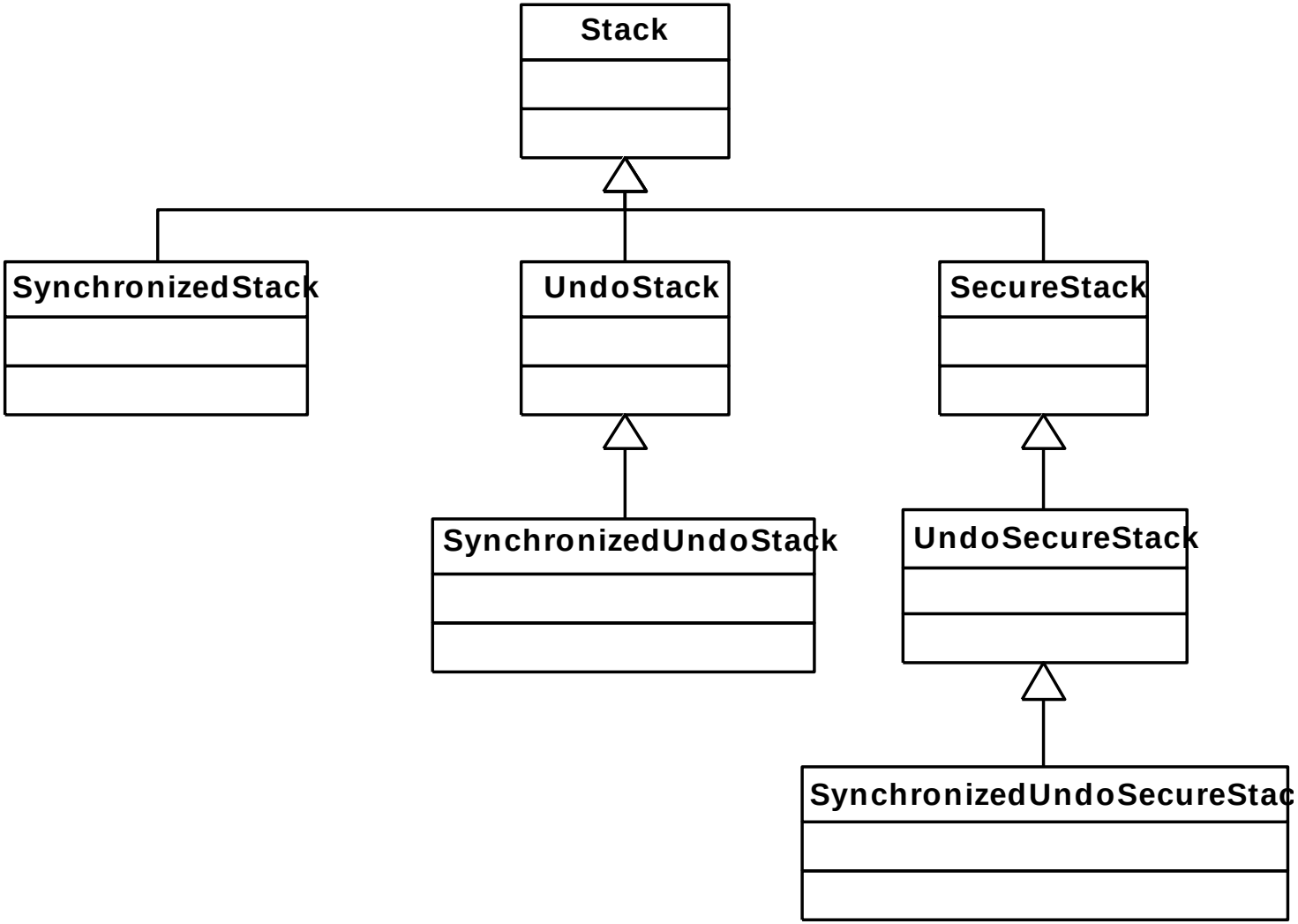
STRATEGY PATTERN



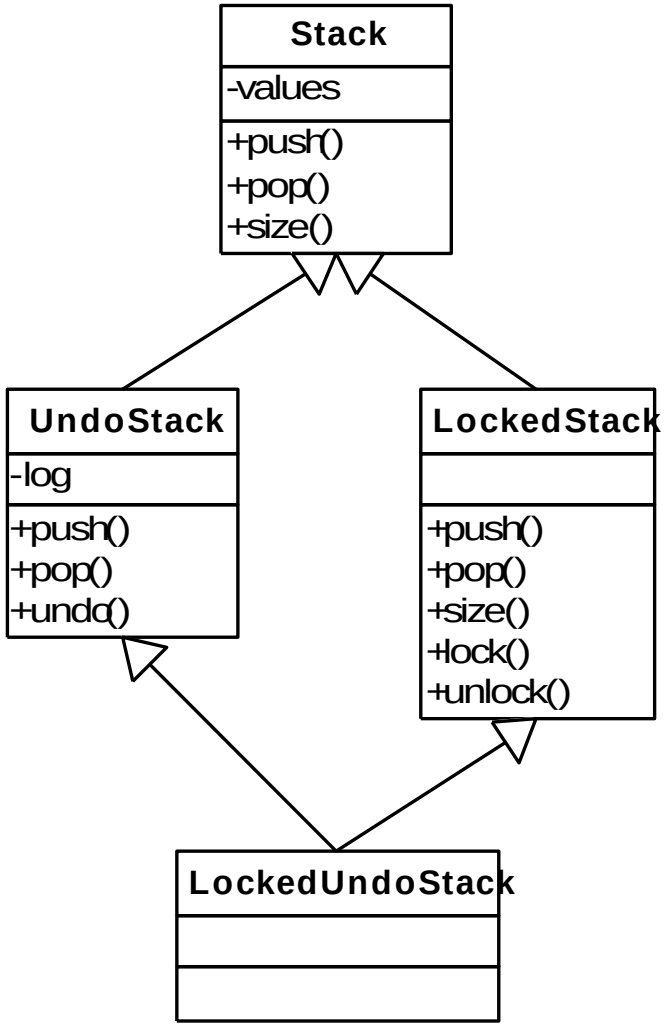
REUSE WITH INHERITANCE


Stack
UndoStack
SecureStack
SynchronizedStack


Stack
SecureStack
SynchronizedStack
UndoStack


Stack
SynchronizedStack
UndoStack
SecureStack
SynchronizedUndoStack
UndoSecureStack
SynchronizedUndoSecureStac

DIAMANT PROBLEM



What happens:

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

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

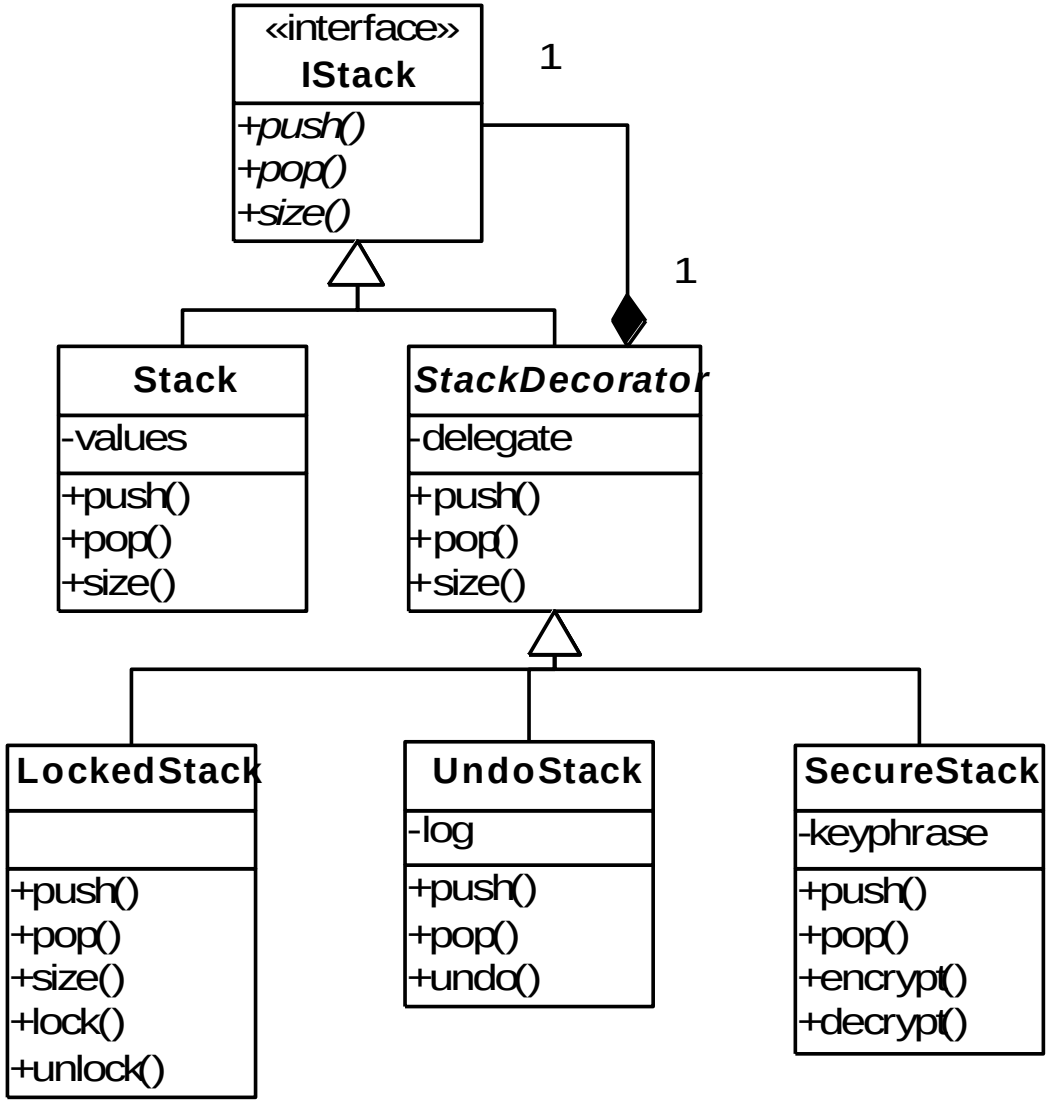
DELEGATION

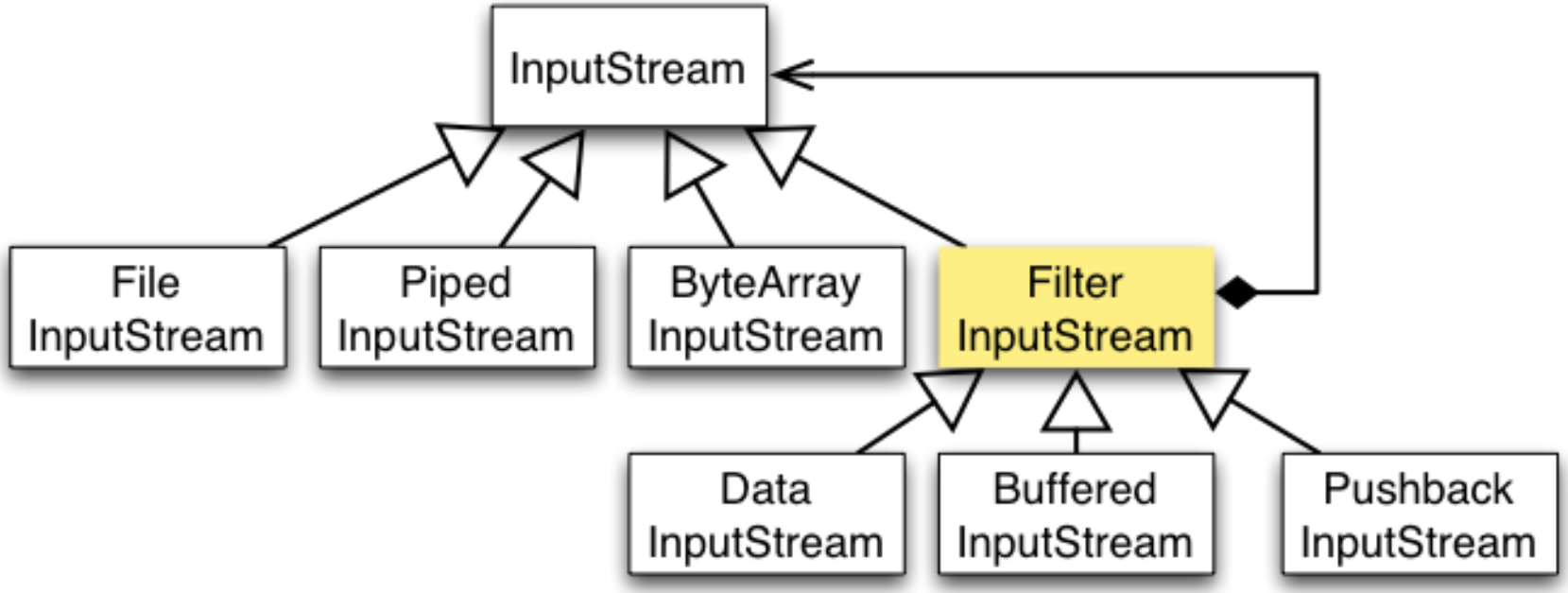
```
class LockedStack implements IStack {
  final IStack _delegate;
  public LockedStack(IStack delegate) {
    this._delegate = delegate;
  }
  private void lock() { /* ... /* }
  private void unlock() { /* ... /* }
  public void push(Object o) {
    lock();
    _delegate.push(o);
    unlock();
  }
  public Object pop() {
    lock();
    Object result = _delegate.pop();
    unlock();
    return result;
  }
  public int size() {
    return _delegate.size();
  }
}
```

```
class UndoStack implements IStack {
  final IStack _delegate;
  public UndoStack(IStack delegate) {
    this._delegate = delegate;
  }
  public void undo() { /* ... /* }
  public void push(Object o) {
    remember();
    _delegate.push(o);
  }
  public Object pop() {
    remember();
    return _delegate.pop();
  }
  public int size() {
    return _delegate.size();
  }
}
```

```
Main:
IStack stack = new UndoStack(
    new LockedStack(new Stack()));
```

DECORATOR PATTERN




InputStream
File InputStream
Piped InputStream
ByteArray InputStream
Filter InputStream
Data InputStream
Buffered InputStream
Pushback InputStream

DISCUSSION TRADEOFFS

Modularity?

Traceability?

Effort?

Granularity?

Uniformity?

FRAMEWORKS

FRAMEWORK

A *framework* is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuse at a larger granularity than classes. A framework is open for extension at explicit *hot spots*.

(Johnson and Foote 1988)

TERMINOLOGY: LIBRARIES

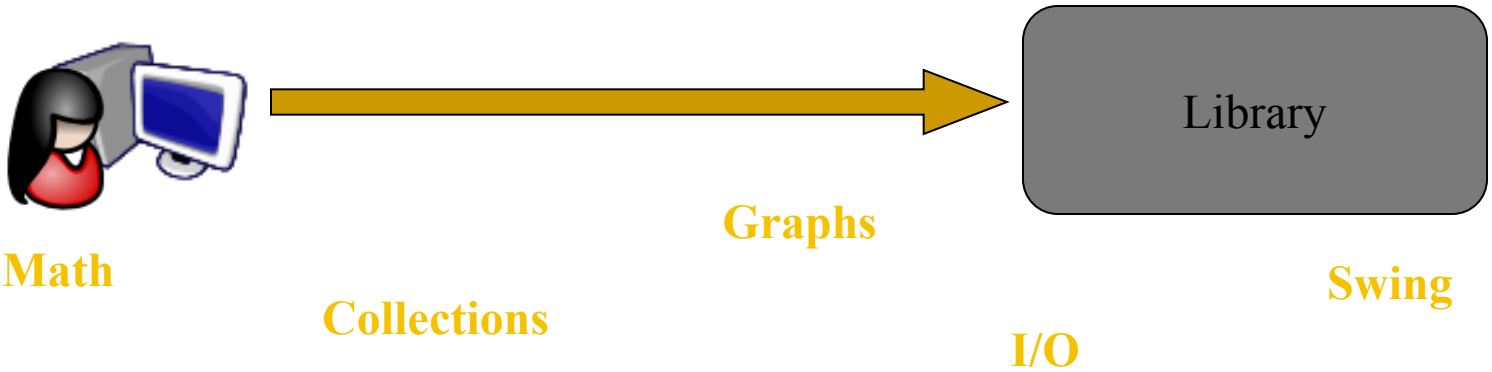
Library: A set of classes and methods that provide reusable functionality

Client calls library to do some task

Client controls

- System structure
- Control flow

The library executes a function and returns data



TERMINOLOGY: FRAMEWORKS

Framework: Reusable skeleton code that can be customized into an application

Framework controls

- Program structure
- Control flow

Framework calls back into client code

- The Hollywood principle: “Don’t call us. We’ll call you.”

```
public MyWidget extends JContainer {

ublic MyWidget(int param) {// setup
internals, without rendering
}

// render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());                }
}
```

your code

Framework

Eclipse

Firefox

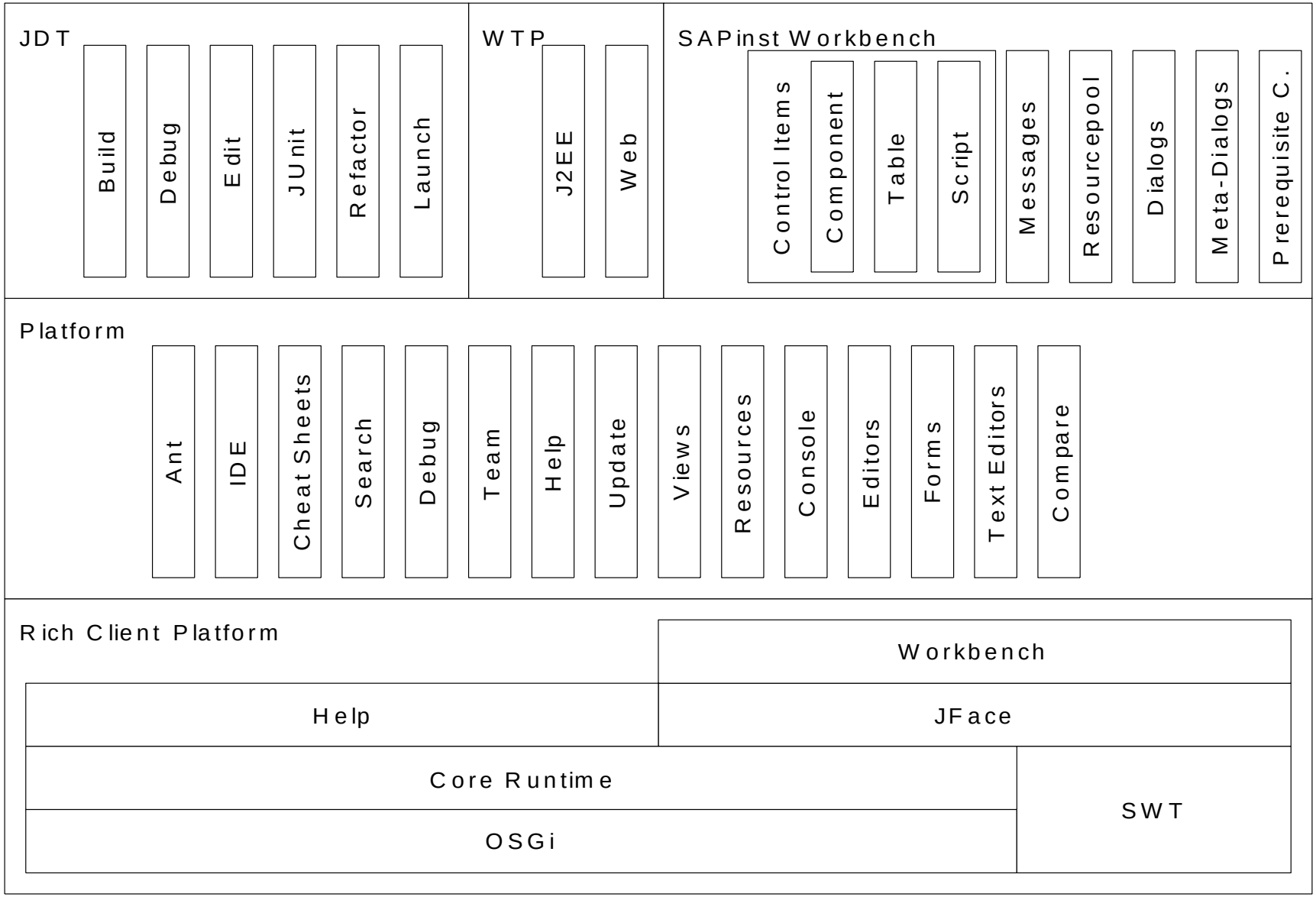
Swing

Applet

Spring

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

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


JDT
Build
Debug
Edit
JUnit
Refactor
Launch
WTP
J2EE
Web
SAPinst Workbench
Control Items
Component
Table
Script
Messages
Resourcepool
Dialogs
Meta-Dialogs
Prerequisite C.
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

A CALCULATOR EXAMPLE (WITHOUT A FRAMEWORK)


```
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(/* calculate some stuff */);
                this.setContentPane(contentPane);
                this.pack();
                this.setLocation(100, 100);
                this.setTitle("My Great Calculator");
                // impl. for closing the window
        }
}
```

A SIMPLE EXAMPLE FRAMEWORK

Consider a family of programs consisting of buttons and text fields only:

What source code might be shared?

A SIMPLE EXAMPLE FRAMEWORK

Consider a family of programs consisting of buttons and text fields only:

What source code might be shared?

- Main method
- Initialization of GUI
- Layout
- Closing the window
- …

A CALCULATOR EXAMPLE


```
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(/* calculate some stuff */);
                this.setContentPane(contentPane);
                this.pack();
                this.setLocation(100, 100);
                this.setTitle("My Great Calculator");
                // impl. for closing the window
        }
}
```

A SIMPLE EXAMPLE FRAMEWORK

```
public abstract class Application extends JFrame {
        protected abstract String getApplicationTitle();
        protected abstract String getButtonText();
        protected String getInititalText() {return "";}
        protected void buttonClicked() { }
        private JTextField textfield;
        public Application() { init(); }
        protected void init() {
                JPanel contentPane = new JPanel(new BorderLayout());
                contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
                JButton button = new JButton();
                button.setText(getButtonText());
                contentPane.add(button, BorderLayout.EAST);
                textfield = new JTextField("");
                textfield.setText(getInititalText());
                textfield.setPreferredSize(new Dimension(200, 20));
                contentPane.add(textfield, BorderLayout.WEST);
                button.addActionListener(/* … buttonClicked(); … */);
                this.setContentPane(contentPane);
                this.pack();
                this.setLocation(100, 100);
                this.setTitle(getApplicationTitle());
                // impl. for closing the window
        }
    protected String getInput() { return textfield.getText();}
}
```

USING THE SIMPLE EXAMPLE FRAMEWORK

```
public abstract class Application extends JFrame {
        protected abstract String getApplicationTitle();
        protected abstract String getButtonText();
        protected String getInititalText() {return "";}
        protected void buttonClicked() { }
        private JTextField textfield;
        pub
        prot
```

```
public class Calculator extends Application {
        protected String getButtonText() { return "calculate"; }
        protected String getInititalText() { return "(10 – 3) * 6"; }
        protected void buttonClicked() {
                JOptionPane.showMessageDialog(this, "The result of "+getInput()+
                        " is "+calculate(getInput())); }
        protected String getApplicationTitle() { return "My Great Calculator"; }
        public static void main(String[] args) {
                new Calculator().setVisible(true);
        }
}
```

```
        this.setContentPane(contentPane);
        this.pack();
        this.setLocation(100, 100);
        this.setTitle(getApplicationTitle());
        // impl. for closing the window
    }
    protected String getInput() { return textfield.getText();}
}
```

USING THE SIMPLE EXAMPLE FRAMEWORK (AGAIN)

```
public abstract class Application extends JFrame {
        protected abstract String getApplicationTitle();
        protected abstract String getButtonText();
        protected String getInititalText() {return "";}
        protected void buttonClicked() { }
        private JTextField textfield;
        pub
        prot
    }
    protected
}
```

```
public class Calculator extends Application {
        protected String getButtonText() { return "calculate"; }
        protected String getInititalText() { return "(10 – 3) * 6"; }
        protected void buttonClicked() {
                JOptionPane.showMessageDialog(this, "The result of "+getInput()+
                        " is "+calculate(getInput())); }
        protected String getApplicationTitle() { return "My Great Calculator"; }
        public static void main(String[] args) {
                new Calculator().setVisible(true);
        }
}
```

```
public class Ping extends Application {
        protected String getButtonText() { return "ping"; }
        protected String getInititalText() { return "127.0.0.1"; }
        protected void buttonClicked() {  /* … */  }
        protected String getApplicationTitle() { return "Ping Anything"; }
        public static void main(String[] args) {
                new Ping().setVisible(true);
        }
}
```

AN EXAMPLE BLACKBOX FRAMEWORK

```
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 ap
}
```

AN EXAMPLE BLACKBOX FRAMEWORK

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

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

                contentP
                textfield
                if (plugin

                textfield.
                contentP
                if (plugin

                this.setC
                …
        }
    public String getInput()
}
```

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

AN ASIDE: PLUGINS COULD BE REUSABLE, TOO…

```
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 Borde
                contentPane.setBorder(new BevelBorder(Bev
                JButton button = new JButton();
                if (plugin != null)
                        button.setText(plugin.getButtonTex
                else
                        button.setText("ok");
                contentPane.add(button, BorderLayout.EAST);
                textfield = new JTextField("");
                if (plugin

                textfield.
                contentP
                if (plugin

                this.setC
                …
        }
        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); }}
```

WHITEBOX VS. BLACKBOX FRAMEWORK SUMMARY

Whitebox frameworks use subclassing

- Allows to extend every nonprivate method
- Need to understand implementation of superclass
- Only one extension at a time
- Compiled together
- Often so-called developer frameworks

Blackbox frameworks use composition

- Allows to extend only functionality exposed in interface
- Only need to understand the interface
- Multiple plugins
- Often provides more modularity
- Separate deployment possible (.jar, .dll, …)
- Often so-called end-user frameworks, platforms

THE COST OF CHANGING A FRAMEWORK

```
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 BorderLa
                contentPane.setBorder(new BevelBorder(BevelB
                JButton button = new JButton();
                if (plugin != null)
                        button.setText(plugin.getButtonText()
                else
                        button.setText("ok");
                contentPane.add(button, BorderLayout.EAST);
                textfield
                if (plugin

                textfield.
                contentP
                if (plugin

                this.setC
                …
        }
    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); }}
```

FRAMEWORK DESIGN CONSIDERATIONS

Once designed there is little opportunity for change

Key decision: Separating common parts from variable parts

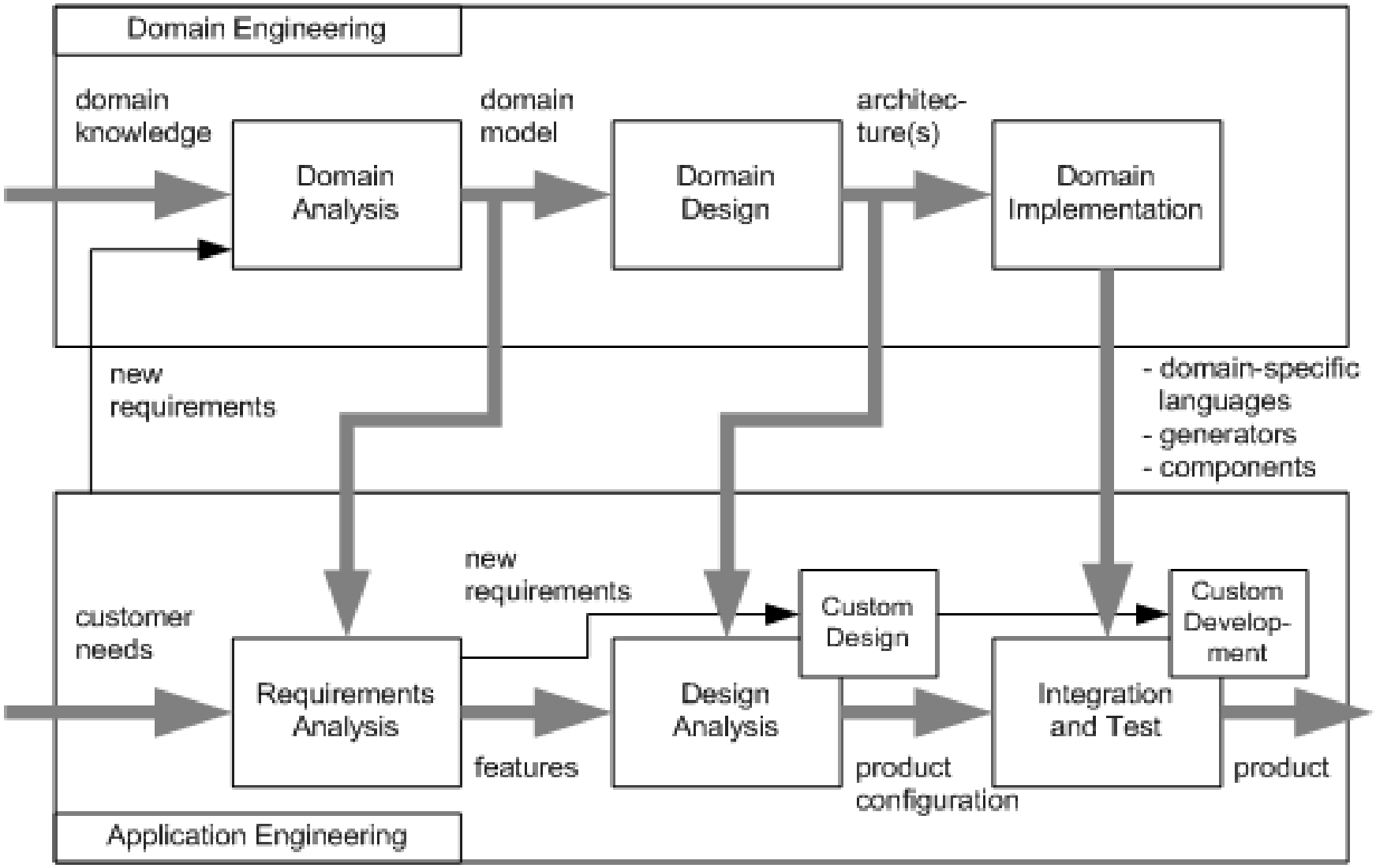
- Identify hot spots vs. cold spots

Possible problems:

- Too few extension points: Limited to a narrow class of users
- Too many extension points: Hard to learn, slow
- Too generic: Little reuse value

The golden rule of framework design:

- Writing a plugin/extension should NOT require modifying the framework source code


Domain Engineering
domain knowledge
Domain Analysis
domain model
Domain Design
architec-ture(s)
Domain Implementation
new requirements
- domain-specific languages
- generators
- components
new requirements
customer needs
Requirements Analysis
features
Design Analysis
Custom Design
product configuration
Integration and Test
Custom Develop-ment
product
Application Engineering

DISCUSSION TRADEOFFS

Binding time?

Modularity?

Traceability?

Effort?

Granularity?

Uniformity?

PLATFORM/SOFTWARE ECOSYSTEM

Hardware/software environment (frameworks, libraries) for building applications

Ecosystem: Interaction of multiple parties on a platform, third-party contributions, co-dependencies, …

- Typically describes more business-related and social aspects

COMPONENTS

COMPONENT

A *software component* is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

(Szyperski 1997)

THE USE VS. REUSE DILEMMA

Large rich components are very useful, but rarely fit a specific need

Small or extremely generic components often fit a specific need, but provide little benefit

“maximizing reuse minimizes use”

C. Szyperski

COMPONENT MARKETS

Best of breed

Business model

Components vs classes

JavaEE, CORBA, COM+/DCOM, OSGi

Service-oriented architectures

COMPONENTS IN PRODUCT LINES

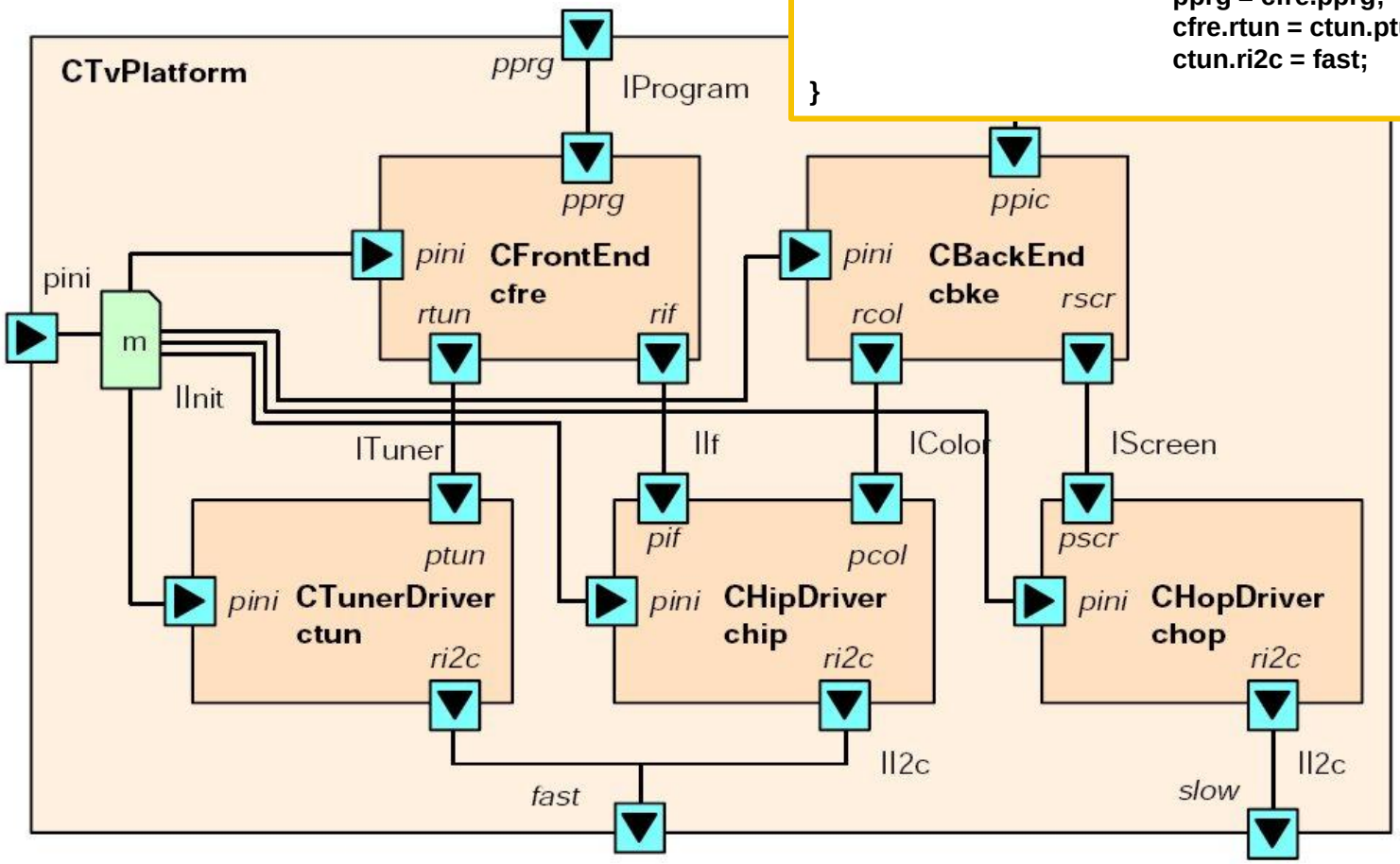
Composition from components

"glue code" to compose

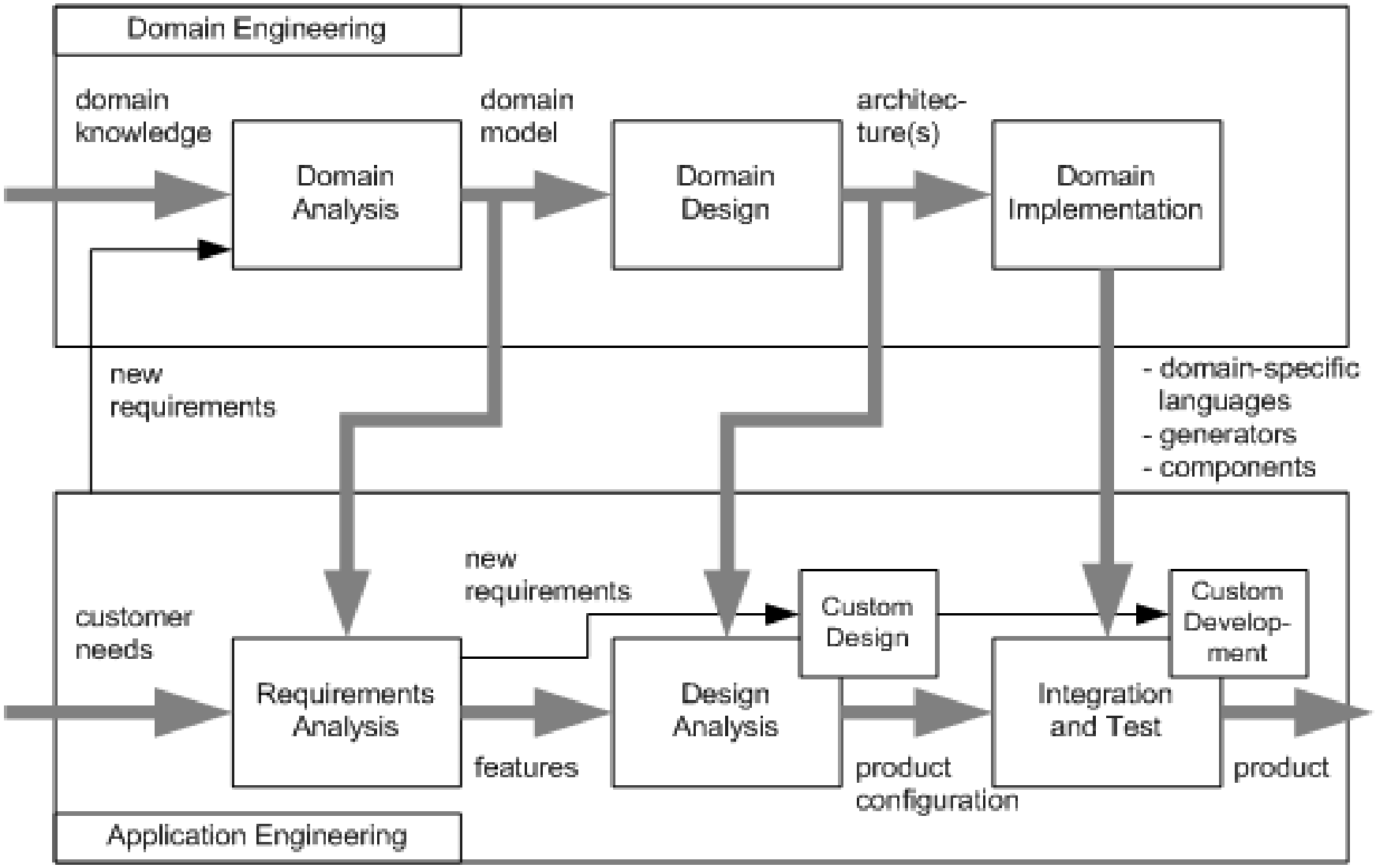
Composition per application (development with reuse)

KOALA (PHILIPS)

```
component CTvPlatform
{
        provides IProgram pprg;
        requires II2c slow, fast;
        contains
                component CFrontEnd cfre;
                component CTunerDriver ctun;
        connects
                pprg = cfre.pprg;
                cfre.rtun = ctun.ptun;
                ctun.ri2c = fast;
}
```



Van Ommering, R., Van Der Linden, F., Kramer, J., & Magee, J. (2000). The Koala component model for consumer electronics software. *Computer*, *33*(3), 78-85.


Domain Engineering
domain knowledge
Domain Analysis
domain model
Domain Design
architec-ture(s)
Domain Implementation
new requirements
- domain-specific languages
- generators
- components
new requirements
customer needs
Requirements Analysis
features
Design Analysis
Custom Design
product configuration
Integration and Test
Custom Develop-ment
product
Application Engineering

DISCUSSION TRADEOFFS

Modularity?

Traceability?

Effort?

Granularity?

Uniformity?

PREPLANNING PROBLEM

PREPLANNING PROBLEM

Need to plan extension points before building extensions

(Noninvasive changes)

Opportunity costs during evolution?

FURTHER READING

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.
Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1997 – Chapters 3, 4

Van Ommering, Rob. "Building product populations with software components." Proceedings of the 24th international conference on Software engineering. ACM, 2002.

Bosch, Jan. "From software product lines to software ecosystems." Proceedings of the 13th International Software Product Line Conference. 2009.