

Probabilistic Detection of GoF Design Patterns

Details of performing the proposed method for some of the Singleton, Factory Method, Composite, Decorator, Command, Visitor, Adapter, Observer, and State/Strategy patterns on five open-source projects, namely JUnit v3.7, JHotDraw v5.1, QuickUML 2001, JRefactory v2.6.24, and MapperXML v1.9.7 are presented as follows.

- Details of Composite pattern (Standard Composite) calculations on QuickUML2001 source code.

package diagram.tool;
import java.util.Iterator;
import java.util.Vector;
import diagram.Diagram;
public class CompositeTool implements Tool { **Implementation**
 private Vector tools = new Vector();
 Implementation
 public void addToolListener(ToolListener l) {
 Overridden Method
 for(Iterator i = tools.iterator(); i.hasNext();)
 ((Tool)i.next()).addToolListener(l);
 }
 Implementation
 public void removeToolListener(ToolListener l) {
 Overridden Method
 for(Iterator i = tools.iterator(); i.hasNext();)
 ((Tool)i.next()).removeToolListener(l);
 }
 Implementation
 public void install(Diagram diagram) {
 Overridden Method
 for(Iterator i = tools.iterator(); i.hasNext();)
 ((Tool)i.next()).install(diagram);
 }
 Implementation
 public void uninstall(Diagram diagram) {
 Overridden Method
 for(Iterator i = tools.iterator(); i.hasNext();)
 ((Tool)i.next()).uninstall(diagram);
 }
 public void add(Tool tool) {
 if(!tools.contains(tool))
 tools.add(tool);
 }
 public void remove(Tool tool) {
 tools.remove(tool);
 }

Composite

package diagram.tool;
import diagram.Diagram;
public interface Tool { **Interface**
 public void addToolListener(ToolListener l);
 Propagation method
 public void removeToolListener(ToolListener l);
 Propagation method
 public void install(Diagram diagram);
 Propagation method
 public void uninstall(Diagram diagram);
 Propagation method
}

Component

- Details of Composite pattern (Standard Composite) calculations on JRefactory v2.6.24 source code.

```

package org.acm.seguin.parser.ast;
import org.acm.seguin.pretty.ModifierHolder;

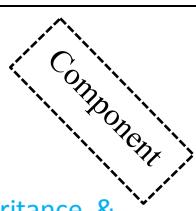
.

.

public class ASTConstructorDeclaration extends SimpleNode implements JavaDocable { Inheritance &
Implementation
    private String name;
    private ModifierHolder modifiers;
    private JavaDocableImpl jdi; Association
    public ASTConstructorDeclaration(int id) { Constructor
        super(id);
        modifiers = new ModifierHolder();
        jdi = new JavaDocableImpl();
    }
    public ASTConstructorDeclaration(JavaParser p, int id) { Constructor
        super(p, id);
        modifiers = new ModifierHolder();
        jdi = new JavaDocableImpl();
    }
    .
    .
    .

    public booleanisRequired() { ForceJavadocComments fjc = new ForceJavadocComments();
        return jdi.isRequired() && method call
            fjc.isJavaDocRequired("method", modifiers); }
    public void addModifier(String modifier) {
        modifiers.add(modifier); }

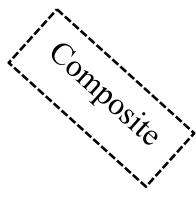
```



```

package org.acm.seguin.pretty;
public class JavaDocableImpl implements JavaDocable { Implementation
    private Vector docs;
    private boolean printed;
    public JavaDocableImpl(){
        docs = new Vector();
        printed = false;}
    public booleanisRequired(){ overridden method
        return !printed; }
    .
    .
    .

```



- Details of Adapter pattern (Standard Adapter) calculations on MapperXML v1.9.7 source code.

<pre> package com.taursys.dom; import org.w3c.dom.Element; import org.w3c.dom.Document; import java.io.OutputStream; import java.io.Writer; public interface DocumentAdapter { public void write(OutputStream stream); public void write(Writer writer); } Target Function, Calling Method public void setElementText(String elementId, String value); public void setAttributeText(String elementId, String attribute, String value); public Element getElementById(String elementId); public Document getDocument(); } </pre>	<p><i>Target</i></p>	<pre> package com.taursys.dom; import java.io.OutputStream; package com.taursys.dom; import java.io.OutputStream; import java.io.Writer; import org.w3c.dom.Document; public abstract class AbstractWriter { public abstract void write(Document doc, OutputStream stream); public abstract void write(Document doc, Writer writer); } </pre>	<p><i>Adaptee</i></p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------

<pre> package com.taursys.dom; import org.w3c.dom.Element; . . . public class DOM_1_20000929_DocumentAdapter implements DocumentAdapter { private Document doc; private Hashtable identifierMap; private String defaultIdentifier = "id"; private AbstractWriter xmlWriter; public DOM_1_20000929_DocumentAdapter(Document newDoc) { setDocument(newDoc); xmlWriter = createDefaultWriter(); } public void write(OutputStream stream) {xmlWriter.write(doc, stream);} public void write(Writer writer) {xmlWriter.write(doc, writer);} public Element getElementById(String elementId) {if (elementId != null) return (Element)identifierMap.get(elementId); else return null;} public void setElementText(String elementId, String value) { setElementText(getElementById(elementId), value);} public void setAttributeText(String elementId, String attribute, String value) { Element element = getElementById(elementId); if (element != null) {if (value == null) value = "";} } } </pre>	<p><i>Adapter</i></p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------

- Details of Adapter pattern (Standard Adapter) calculations on QuickUML 2001 source code.

```

package diagram;
import java.awt.Component;
import javax.swing.JTextField;
public class DefaultFigureEditor extends DefaultCellEditor
    implements FigureEditor {Inheritance (A child Class)
    public DefaultFigureEditor() {Constructor
        this(new JTextField());}
    public DefaultFigureEditor(JTextField comp) {
        super(comp);
        setClickCountToStart(2);}
    public Component getFigureEditorComponent(Diagram diagram, Figure figure, boolean isSelected) {
        JTextField comp = ((JTextField)getComponent());
        Object label = diagram.getModel().getValue(figure);
        comp.setText(label != null ? label.toString() : "");
        comp.setOpaque(true);
        comp.setForeground(diagram.getForeground());
        comp.setBorder(BorderFactory.createLineBorder(diagram.getBackground().darker()));
        return comp;}
    public Object getCellEditorValue() {Target Function, Calling Method
        return ((JTextField)getComponent()).getText();}
    public Rectangle2D getDecoratedBounds(Diagram diagram, Figure figure, Rectangle2D rcBounds) {
        FigureRenderer renderer = diagram.getFigureRenderer(figure.getClass());
        return renderer.getDecoratedBounds(diagram, figure, rcBounds);}}
```

```

package uml.diagram;
import java.awt.BorderLayout;
public class DefaultLabelRenderer
{
    .
    .
    .
    UIManager.put("note.foreground", Color.black);
    UIManager.put("note.border", new NoteBorder());
    public NoteComponent() {setLayout(new BorderLayout());
        text.setBorder(null);
        text.setMargin(margin);
        add(text);
        setUI(noteUI);}
    public void setText(String s) {text.setText(s);}
    public String getText() {return text.getText();}}Adaptee
Adaptee Function, Called method
```

```

package uml.diagram;
import java.awt.Component;
public class diagram.DefaultLinkEditor
{
    .
    .
    .
    public Component
        getFigureEditorComponent(Diagram diagram, Figure
        figure, boolean isSelected) {
            noteComponent =
            (NoteComponent)renderer.getUserComponent();
            return editorComponent;}
    public Object getCellEditorValue() {Request
        Redefinition in Adapter
        return noteComponent.getText();}}
```

Adapter

- Details of Factory Method pattern (variant type: Multiple Product Types inside Factory Class) calculations on JUnit v3.7 source code.

<pre> package junit.samples.money; class Money implements IMoney { Implementation private int fAmount; private String fCurrency; public Money(int amount, String currency) { fAmount= amount; fCurrency= currency; } public IMoney add(IMoney m) { return m.addMoney(this);} public IMoney addMoney(Money m) { overridden method if (m.currency().equals(currency())) return new Money(amount()+ m.amount(), currency()); return new MoneyBag(this, m);} public IMoney addMoneyBag(MoneyBag s) { overridden method & Association return s.addMoney(this); } method call public int amount() {return fAmount; } . . . public boolean isZero() { return amount() == 0; } public IMoney multiply(int factor) { overridden method return new Money(amount()*factor, currency()); } public IMoney negate() { return new Money(-amount(), currency()); } Return Instance public IMoney subtract(IMoney m) { return add(m.negate()); } method call public String toString() { StringBuffer buffer = new StringBuffer(); buffer.append("["+amount()+" "+currency()+"]"); return buffer.toString(); } } </pre>	<pre> class Money { Implementation . . . public Money(int amount, String currency) { fAmount= amount; fCurrency= currency; } public IMoney add(IMoney m) { return m.addMoney(this);} public IMoney addMoney(Money m) { overridden method if (m.currency().equals(currency())) return new Money(amount()+ m.amount(), currency()); return new MoneyBag(this, m);} public IMoney addMoneyBag(MoneyBag s) { overridden method & Association return s.addMoney(this); } method call public int amount() {return fAmount; } . . . public boolean isZero() { return amount() == 0; } public IMoney multiply(int factor) { overridden method return new Money(amount()*factor, currency()); } public IMoney negate() { return new Money(-amount(), currency()); } Return Instance public IMoney subtract(IMoney m) { return add(m.negate()); } method call public String toString() { StringBuffer buffer = new StringBuffer(); buffer.append("["+amount()+" "+currency()+"]"); return buffer.toString(); } } </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- Details of Factory Method pattern (Standard Factory Method) calculations on JUnit v3.7 source code.

```
package junit.samples.money;
interface IMoney { Interface
    public abstract IMoney add(IMoney m);
    IMoney addMoney(Money m); Propagation Method
    IMoney addMoneyBag(MoneyBag s); Propagation Method
    public abstract boolean isZero();
    public abstract IMoney multiply(int factor); Propagation Method
    public abstract IMoney negate();
    public abstract IMoney subtract(IMoney m);
}
```

Creator

```
package junit.samples.money;
import java.util.*;
class MoneyBag implements IMoney { Implementation
    private Vector fMonies= new Vector(5);
    private MoneyBag() {}
    MoneyBag(Money bag[]) {
        for (int i= 0; i < bag.length; i++) {
            if (!bag[i].isZero())
                appendMoney(bag[i]);}
    }
    MoneyBag(Money m1, Money m2) { Constructor
        appendMoney(m1);
        appendMoney(m2); }
    MoneyBag(Money m, MoneyBag bag) {
        appendMoney(m);
        appendBag(bag); }
    MoneyBag(MoneyBag m1, MoneyBag m2) {
        appendBag(m1);
        appendBag(m2); }
    public IMoney add(IMoney m) {
        return m.addMoneyBag(this); }
    public IMoney addMoney(Money m) { overridden method
        return (new MoneyBag(m, this)).simplify(); }
    public IMoney addMoneyBag(MoneyBag s) { overridden method
        return (new MoneyBag(s, this)).simplify(); }

    public IMoney multiply(int factor) { overridden method
        MoneyBag result= new MoneyBag(); Return Instance
        .
        .
        .
    }
}
```

Concrete Creator

- Details of Observer pattern (Standard Observer) calculations on JHotDraw V5.1 source code.

```

package CH.ifaf.draw.standard;
import java.awt.Point;
.

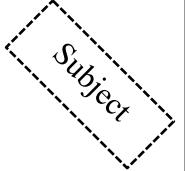
.

public class StandardDrawing extends CompositeFigure implements Drawing { Inheritance & Implementation
    private transient Vector fListeners;
    private transient Thread fDrawingLockHolder = null;
    private static final long serialVersionUID ;
    private int drawingSerializedDataVersion = 1;
    public StandardDrawing() { Constructor
        super();
        fListeners = new Vector(2);
    }
    public void addDrawingChangeListener(DrawingChangeListener listener) { Association
        fListeners.addElement(listener);
    }
    public void removeDrawingChangeListener(DrawingChangeListener listener) { Association
        fListeners.removeElement(listener);
    }
    public Enumeration drawingChangeListeners() { Association
        return fListeners.elements();
    }
    public synchronized Figure remove(Figure figure) {
        .

        .

        public void figureRequestUpdate(FigureChangeEvent e) {
            if (fListeners != null) {
                for (int i = 0; i < fListeners.size(); i++) {

```



```

package CH.ifaf.draw.framework;
import java.util.EventListener;
public interface DrawingChangeListener extends EventListener { Interface & Inheritance
    public void drawingInvalidated(DrawingChangeEvent e);
    public void drawingRequestUpdate(DrawingChangeEvent e);}

```



- Details of Observer pattern (Standard Observer) calculations on JUnit v3.7 source code.

```

package junit.framework;
import java.util.Enumeration;
import java.util.Vector;
public class TestResult extends Object { Inheritance
    protected Vector fFailures;
    protected Vector fErrors;
    protected Vector fListeners;
    protected int fRunTests;
    private boolean fStop;
    public TestResult() {
        .
        .
        .

    private synchronized Vector cloneListeners() {
        return (Vector)fListeners.clone();
    public void endTest(Test test) { Association
        for (Enumeration e= cloneListeners().elements(); e.hasMoreElements(); ) {
            ((TestListener)e.nextElement()).endTest(test);}
    public synchronized int errorCount() {
        return fErrors.size();}
    public synchronized Enumeration errors() {
        return fErrors.elements();}
    public synchronized int failureCount() {
        return fFailures.size();}
    public synchronized Enumeration failures() {
        return fFailures.elements();}
    protected void run(final TestCase test) { Association & Notify function
        startTest(test); }
        .
        .
        .
    }
}

```

Subject

```

package junit.framework;
public interface Test { Interface
    public abstract void run(TestResult result);}

```

Observer

- Details of Singleton pattern (variant type: Eager Instantiation) calculations on JHotDraw V5.1 source code.

```

package CH.ifaf.draw.framework;

import CH.ifaf.draw.util.*;
import java.awt.*;
import java.util.*;
import java.io.Serializable;

public interface Figure    Public & Interface
    extends Storable, Cloneable, Serializable { Inheritance (Child Class)
    public void moveBy(int dx, int dy);
    public void basicDisplayBox(Point origin, Point corner);
    public void displayBox(Point origin, Point corner);
    public Rectangle displayBox();

    .
    .

    public Figure findFigureInside(int x, int y); Not Private & return an Instances of Class
    public boolean containsPoint(int x, int y);
    public Object clone();
    public void displayBox(Rectangle r);
    public boolean includes(Figure figure);
    public FigureEnumeration decompose(); Association
    public void addToContainer(FigureChangeListener c);
    public void removeFromContainer(FigureChangeListener c);
    public FigureChangeListener listener();

    public void addFigureChangeListener(FigureChangeListener l);
    public void removeFigureChangeListener(FigureChangeListener l); Dependency
    public void release();
    public void invalidate();
    public void willChange();
    public boolean canConnect();
    public Connector connectorAt(int x, int y);
    public void connectorVisibility(boolean isVisible);
    public Insets connectionInsets();
    public Locator connectedTextLocator(Figure text);
    public Object getAttribute(String name);
    public void setAttribute(String name, Object value);
}

```



- Details of Singleton pattern (Standard Singleton) calculations on JRefactory v2.6.24 source code.

```
package org.acm.seguin.ide.command;
import java.awt.BorderLayout;

public class PackageSelectorPanel extends PackageSelectorArea
    implements ActionListener, Saveable, Reloader {
    protected String rootDir = null;
    private HashMap viewList;
    private ButtonPanel buttons;
    private static PackageSelectorPanel mainPanel;
    protected PackageSelectorPanel(String root) Non-p
    {
        super();
        setRootDirectory(root);
        ReloaderSingleton.register(this);
        ReloaderSingleton.reload();
        buttons = new ButtonPanel(this);
        createFrame(); }
```

Inheritance (Child Class)

Singleton

Non-public Constructor

```
public static PackageSelectorPanel getMainPanel(String directory)
```

not private method & return an instance of singleton

```
{ if (mainPanel == null) { if (directory == null) { return null;}}
```

Conditional statement in the body of instance operation

1

6

1

- Details of State/ Strategy pattern (Standard Strategy) calculations on MapperXML v1.9.7 source code.

```

package com.taursys.xml;
import com.taursys.xml.event.RenderException;
import com.taursys.xml.render.TextFieldRenderer;
import com.taursys.xml.event.RenderEvent;
public class TextField extends AbstractField { Inheritance
    .
    .
    .
    renderer = createDefaultRenderer();}
protected TextFieldRenderer createDefaultRenderer() { Association
    return new TextFieldRenderer(this);}
    .
    .
    .

```

Context

```

package com.taursys.xml.render;
import com.taursys.xml.TextField;
.
.
.
public class TextFieldRenderer extends AbstractRenderer { Inheritance
    public TextFieldRenderer() { Constructor
        super(null);}
    public TextFieldRenderer(TextField textField) { Association & Constructor
        super(textField);}
    public void render(TextField textField) throws RenderException { Propagation method
        .
        .
        .
    }

```

State /
Strategy

```

package com.taursys.xml.render;
.
.
.
public class AttributeTextFieldRenderer extends TextFieldRenderer { inheritance
    public AttributeTextFieldRenderer() {}
    public AttributeTextFieldRenderer(TextField textField) {
        super(textField);}
    public void render(TextField textField) throws RenderException { Overridden Method
        .
        .
        .
    }
    public void render() { Overridden Method
        .
        .
        .
    }

```

Concrete State /
Concrete Strategy

- Details of State/ Strategy pattern (Standard Strategy) calculations on QuickUML 2001 source code.

```

package diagram;
import java.awt.Dimension;
.

.

public class Diagram extends JComponent
return selectionModel;}
public FigureRenderer getFigureRenderer(Class itemClass) { Association
for(Object o = null; o == null && itemClass != null; itemClass = itemClass.getSuperclass()) {
if((o = rendererMap.get(itemClass)) != null)
return (FigureRenderer)o;
return null;}
public void setFigureRenderer(Class itemClass, FigureRenderer renderer) {...}
public void setFigureEditor(Class itemClass, FigureEditor editor) {...}

```

Context

```

package diagram;
import java.awt.Component;
import java.awt.geom.Rectangle2D;
public interface FigureRenderer {
public Component getRendererComponent(Diagram diagram, Figure figure, boolean isSelected); Propagation method
public Rectangle2D getDecoratedBounds(Diagram diagram, Figure figure, Rectangle2D rcBounds); Propagation method

```

State /
Strategy

```

package diagram;
import java.awt.Component;
private Diagram lastDiagram = null;
public DefaultFigureRenderer() {
this(null);}
public DefaultFigureRenderer(Component userComponent) {setLayout(null);
setUserComponent(userComponent);}
public Component getRendererComponent(Diagram diagram, Figure figure, boolean isSelected) { Overridden method
.

.

return this;}
public void setUserComponent(Component component) {
if(component != userComponent) {userComponent = component;
.

.

public Figure getFigure() {return lastFigure;}
public Diagram getDiagram() {return lastDiagram;}
public Rectangle2D getDecoratedBounds(Diagram diagram, Figure figure, Rectangle2D rcBounds) { Overridden method
.

.


```

Concrete State /
Concrete Strategy