

The Good, The Bad, and The Ugly

Automated Ways to Better Software

Patrycja Węgrzynowicz
Head of Software R&D Department

About Us

■ Software R&D Department

- Software Development
 - High quality delivered on-time
- Consulting & Management
- Code Auditing
 - Security and quality of code
- Education
 - Advanced trainings on JEE, Hibernate, Spring, code security

■ Clients

- ICANN, .IE Domain Registry, ABW, BN, NDAP, IPN

Domain names and DNS

.PL Registry
RZM for ICANN (**root**)
.IE Registry

Large-Scale

Digital archives
Workflows
Cloud computing, grid,
virtualization

Security

HoneySpider Network,
FISHA, Arakis, SpotSPAM
(with CERT Polska)

Automated Software Engineering

D-CUBED

Agenda

Automated Software Engineering

Introduction and basic techniques in the context of desing and implementation

The Good

Design abstraction
and design hints

The Bad

Detection of program
defects

The Ugly

Detection of bad
practices

Automated Software Engineering

Lots of methodologies have come and gone, paradigms have changed but the requirements are always the same; Make it good and make it fast.

-- Anonymous

Requirements

■ Make it good

- High expectation: Functional, Quality, Performance, Data

■ Make it fast

- Short time to delivery

■ Scale

- Tons of legacy code...
- More and more code being produced...

Software Engineering

Software engineering is concerned with the **analysis, design, implementation, testing, and maintenance** of software systems.

Automated Software Engineering

How to **automate** or partially **automate** software engineering tasks?

Support in Testing and Maintenance

- Code Level
- How to Comprehend Code?
- How to Detect Bugs?
- How to Improve Code?

Program Analysis

Code Analysis

Static Analysis

- Source code level (or bytecode)
- Look for code patterns
- Compare with abstract model

Dynamic Analysis

- Run program
- Feed it inputs
- See what happens



Theorem prover, inference engines, databases, code patterns



**Design
Patterns & Hints**



Bugs



Bad Practices

The Good

It's OK to figure out murder mysteries, but you shouldn't need to figure out code.

You should be able to read it.

-- Steve C. McConnell

Software Comprehension

Real programmers don't comment their code. It was hard to write, it should be hard to understand.

-- Anonymous

- Legacy code
- Millions lines of code
- Lack of documentation
- Maintenance
 - Extension, Modification, Bug fixing
- New developers

Automatic Detection of Design Patterns

■ Design patterns

- General reusable solution to a commonly occurring problem in software design
- Examples: Singleton, Abstract Factory, Visitor

■ Better understanding of code

- Abstract concepts
- Code intent – why, not only how

■ Better documentation

- Generating of documentation
- Verification of documentation (including object-oriented models)

Selected Pattern Detectors

■ PINOT

- Hard-coded static analysis in a search for pattern-specific code blocks
- Open-source for custom compilation
- Nija Shi, UC Davies

■ Ptidej

- Explanation-based constraint programming
- Pre-Commercial
- University of Montréal

■ FUJABA

- Static analysis combined with fuzzy logic
- Open source
- University of Paderborn

■ D3 (D-CUBED)

- Semantic code query system based on a static analysis and SQL
- To be released at the end of June 2009 – open source
- P. Wegrzynowicz, NASK

Singleton – The Simplest Design Pattern

```
public class S {  
    private static S instance;  
    private S() {}  
    public static synchronized S getInstance()  
        if (instance == null) instance = new S();  
    return instance;  
}  
public static S createInstance() {  
    return new S();  
}  
}
```

PINOT Fails

Satisfied to find lazy instantiation block.

Ptidej Fails

Satisfied to find lazy instantiation block.

FUJABA Fails

Only structural constraints.

D-CUBED Succeeds!

Analyses usage semantic!

Singleton – Not That Simple After All...

- Eager Instantiation
- Lazy Instantiation
- Replaceable Instance
- Subclassed Singleton
- Delegated Construction
- Different Placeholder
- Different Access Point
- Limiton

Subclassed Singleton and Delegated Construction

■ JHotDraw

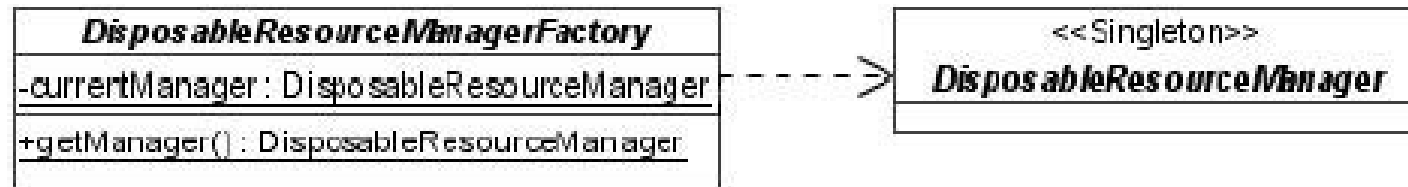
- "design exercise" developed by Erich Gamma and Thomas Eggenschwiler



```
protected static CollectionsFactory determineCollectionsFactory() {
    String jdkVersion = null;
    if (isJDK12()) jdkVersion = "12";
    else jdkVersion = "11";
    return createCollectionsFactory(jdkVersion);
}
```

Different Access Point, Different Placeholder

- JHotDraw
- It is also the example of the Subclassed Singleton



Different Placeholder

■ Initialization of Demand Holder Idiom

- William Pugh – co-author of FindBugs

```
public class Singleton {  
  
    private Singleton() {}  
  
    /**  
     * SingletonHolder is loaded on the first execution of  
     * Singleton getInstance().  
     */  
    private static class SingletonHolder {  
        private final static Singleton INSTANCE = new Singleton();  
    }  
  
    public static Singleton getInstance() {  
        return SingletonHolder.INSTANCE;  
    }  
}
```

Limiton

■ JHotDraw

Alignment
+LEFTS : Alignment
+CENTERS : Alignment
+RIGHTS : Alignment
+TOPS : Alignment
+MIDDLES : Alignment
+BOTTOMS : Alignment
-Alignment(desc : String)

```
public final static Alignment LEFTS = new Alignment("Lefts") {  
    public void moveBy(Figure f, Rectangle anchor) {  
        Rectangle rr = f.displayBox();  
        f.moveBy(anchor.x-rr.x, 0);  
    }  
};  
}
```

Problems in Detection of Design Patterns

■ Lack of Formalization

- Design patterns are abstract concepts
- Textual description
- Canonical form

■ Implementation Variants

- Patterns independently invented
- Implementation crafted to accommodate other constraints
- Multiple patterns combined to solve a problem
- Language-specific code constructs

■ Performance and scalability

The Bad

Bugs are not only likely. They are inevitable.

-- myself

Problem – Bugs

Bug are not only likely. They are **inevitable!**

Problem – Short Time

„Time is so **short**, you can't make a debug...”

-- Scott Adams

Solution – Automated Bug Finders

But you can make use of **automated bug finders**.

Solution – Automated Bug Finders

But you can make use of automated bug finders.

The Bug Finders We Use

■ FindBugs

- General issues
- Daily usage – integrated into our maven builds and the continuous integration server Hudson

■ JLint

- Multithreaded issues
- Specific applications

■ D3 (D-CUBED)

- Security issues
- Code auditing

FindBugs

■ Available at SourceForge

- Standalone application
- Ant task
- Maven plugin
- Eclipse plugin

■ Bugs Categories

- Correctness
- Multithreaded Correctness
- Malicious Code Vulnerability
- Performance
- Security
- Bad Practice and Dodgy

Can you find a bug?

FindBugs can!

(The examples found in JDK1.6.0-b105 by W. Pugh, presented at JavaOne 2007)

```
// com.sun.corba.se.impl.naming.cosnaming.NamingContextImpl  
    if (name != null || name.length() > 0)
```

FindBugs – Null Pointer Exception

■ Null Pointer Exception

- JDK1.6.0-b105
- 109 NPE – 54 serious issues (NPE on valid input)

```
// com.sun.corba.se.impl.naming.cosnaming.NamingContextImpl  
if (name != null || name.length() > 0)
```

```
// com.sun.xml.internal.ws.wSDL.parser.RuntimeWSDLParser  
if (part == null | part.equals(""))
```

```
// sun.awt.x11.ScrollPanePeer  
if (g != null) paintScrollBars(g, colors);  
g.dispose();
```

FindBugs – Infinite Recursive Loop

■ Infinite Recursive Loop

- JDK1.6.0-b13
- 5 infinite recursive loops

```
public String foundType() {  
    return this.foundType();  
}
```

■ Even smart people make dumb mistakes

FindBugs – Bad Method Invocation

- Bad Method Invocation
- Methods whose return value should not be ignored
 - Ignoring the result of `String.trim()` or `String.toLowerCase()` is a bug – strings are immutable in Java

```
// com.sun.rowset.CachedRowSetImpl  
if (type == Types.DECIMAL || type == Types.NUMERIC)  
    ((java.math.BigDecimal)x).setScale(scale);
```

JLint

■ Available at SourceForge

- Binaries available for Windows
- Custom compilation for other platforms

■ Bugs Categories

- Multithreaded issues
- Bugs, inconsistencies and synchronization problems by doing data flow analysis and building the lock graph
- Finds more multithreaded issues than FindBugs

JLint – Deadlocks

■ Capture-HPC

- A high interaction client honeypot
- Identifies malicious servers by interacting with potentially malicious servers using a dedicated a dedicated virtual machine and observing its system state changes
- Used in our HoneySpider Network project

■ Several serious multithreaded bugs

1. **capture\VirtualMachine.java:95**: Loop 1: invocation of synchronized method `capture/Client.setClientState(capture.CLIENT_STATE)` can cause **deadlock**.
2. **capture\Client.java:400**: Loop 1: invocation of synchronized method `capture/VirtualMachine.setState(capture.VM_STATE)` can cause **deadlock**.
3. **capture\VirtualMachine.java:95**: Loop 2: invocation of synchronized method `capture/Client.setClientState(capture.CLIENT_STATE)` can cause **deadlock**.
4. **capture\Client.java:405**: Loop 2/1: invocation of method `capture/Client.disconnect()` forms **the loop** in class dependency graph.
5. **capture\Client.java:150**: Loop 2: invocation of synchronized method `capture/VirtualMachine.setState(capture.VM_STATE)` can cause **deadlock**.

D-CUBED

■ To be available at SourceForge

- June 2009
- Work-in-progress

■ Bugs Categories

- Security issues
- Information leakage
- Confidentiality
- Authorization

D-CUBED – Security Issue

■ Confidentiality

- Ensuring that information is accessible only to those authorized to have access

■ Example

- Password leakage
- Printed out to standard out as part of logging process
- Thrown in an exception

Bug Finders – Summary

■ FindBugs

- Finds generic issues
- Finds a broad range of issues
- Relatively low number of false positives
- Mature
- Good integration with IDE/builds
- Focus on rather small code blocks

■ JLint and D-CUBED

- More sophisticated
- Specific applications
- Worse integration with IDE/builds
- Focus on code interaction and interprocedural flow

The Ugly

Always code as if the guy who ends up maintaining
your code will be a violent psychopath who knows
where you live.

-- Rick Osborne

Fix Ugly Code As Soon as Possible

Don't wait until you have bug to step through your code.

-- Steve Maguire

■ Bad Practices

- Not necessarily bugs... yet!
- Might cause an unexpected behavior in near future
- Hard to maintain and comprehend by others

FindBugs – Bad Practices

- Equals without corresponding hashCode
 - Violates the contract that equal object have the same hashCodes
- Equals that does not handle nulls
- Potentially unsafe usage of getResource
 - Calling `this.getClass().getResource(...)` could give results other than expected if this class is extended by a class in another package
- Ignored exceptions

Summary

Summary

Use It!

Finders of Bugs and Bad Practices

Keep an Eye on It!

Detectors of Design Patterns and Providers of Design Hints

Q&A

Patrycja Węgrzynowicz
patrycjaw (at) nask.pl