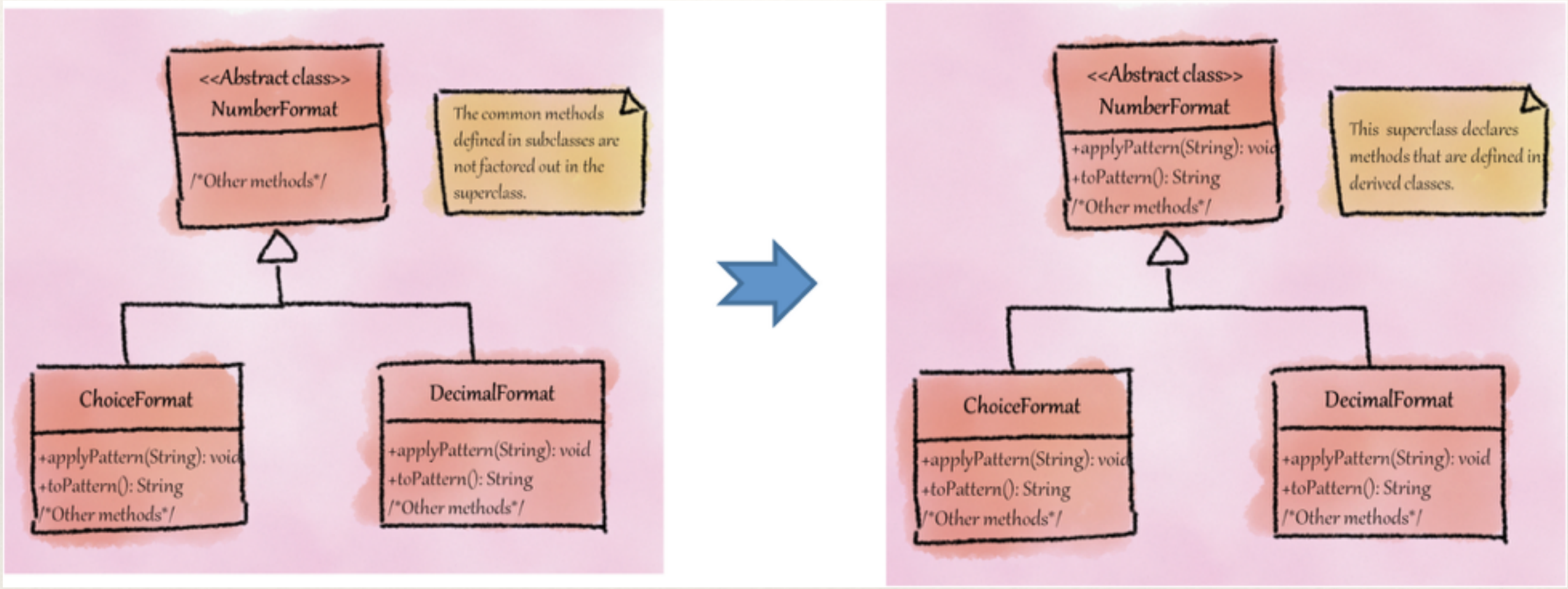
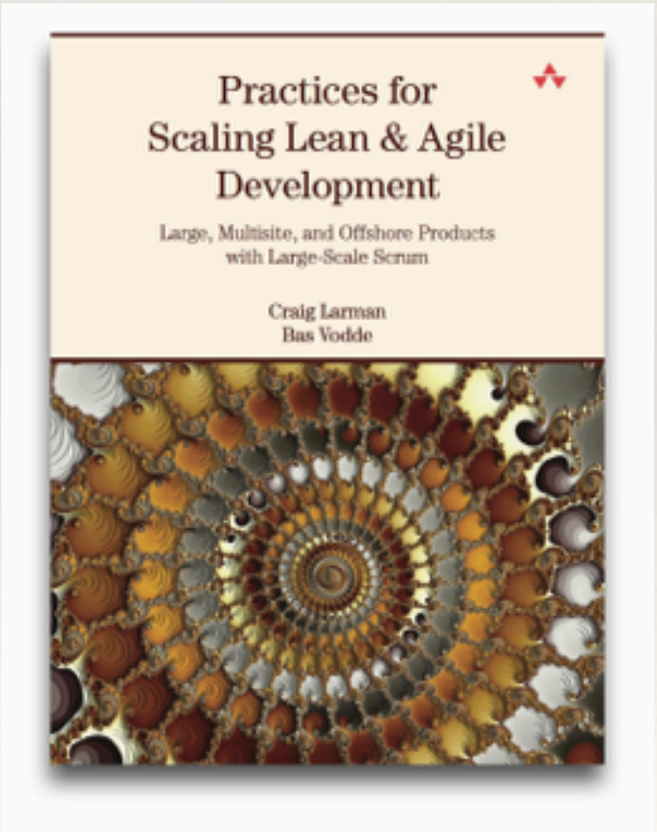
REFACTORING BY EXAMPLE

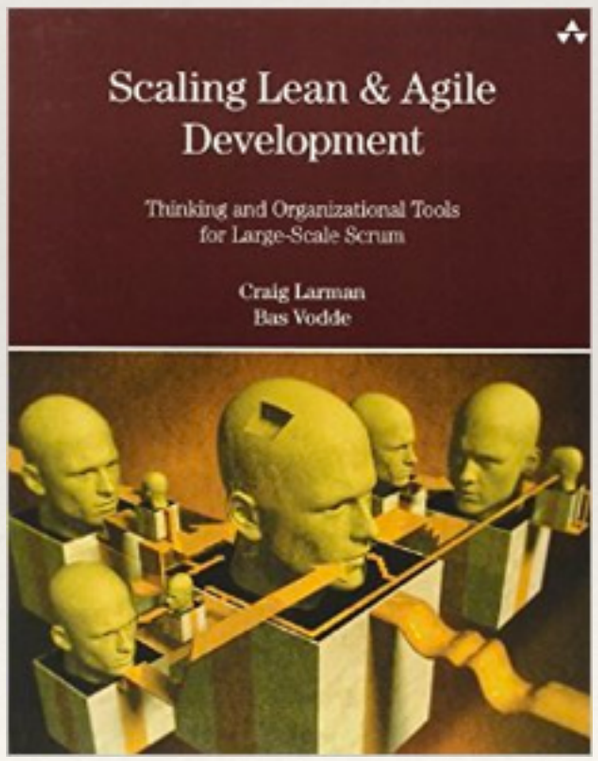


Ganesh Samarthyam
ganesh@codeops.tech

"The critical design tool for software development is a mind well educated in design principles"

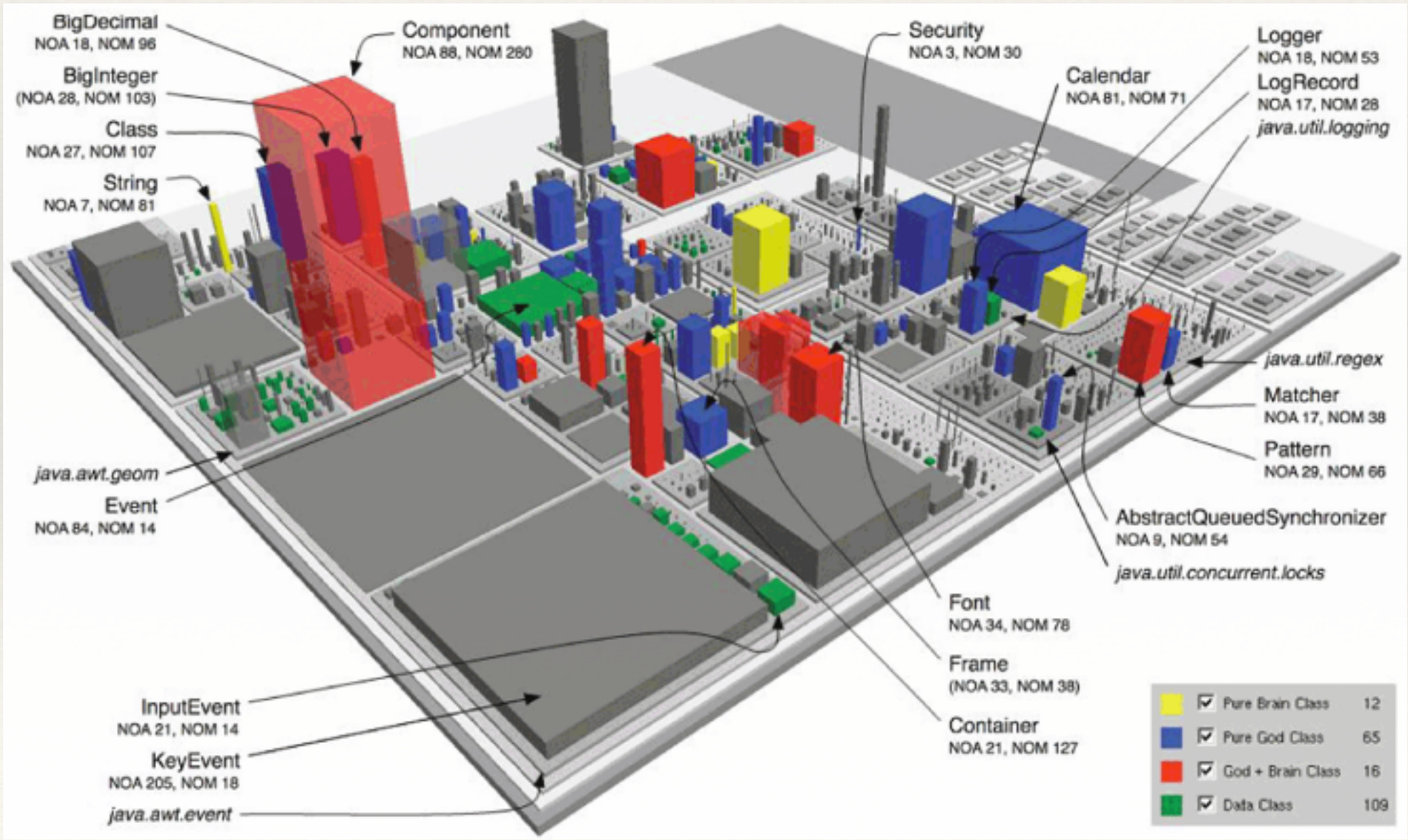
–Craig Larman





Warm-up question #1

What does this tool visualize?



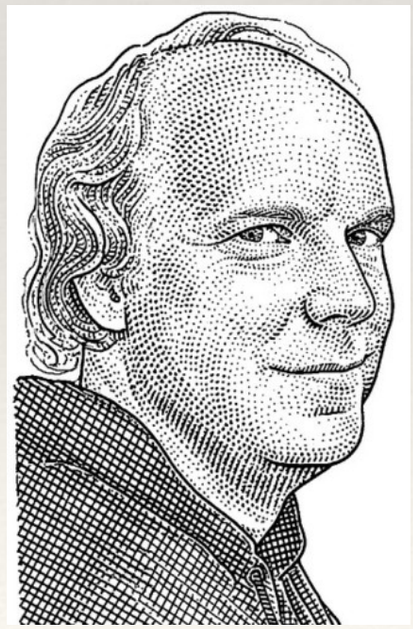
Warm-up question #2

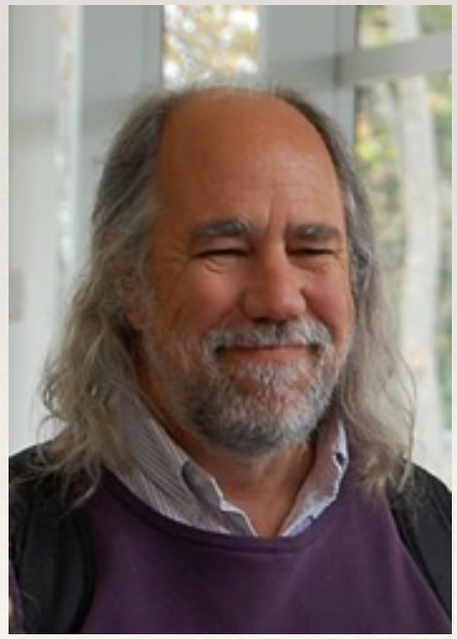
Who coined the term "code smell"?

Hint: He also originated the terms TDD and XP

Warm-up question #3

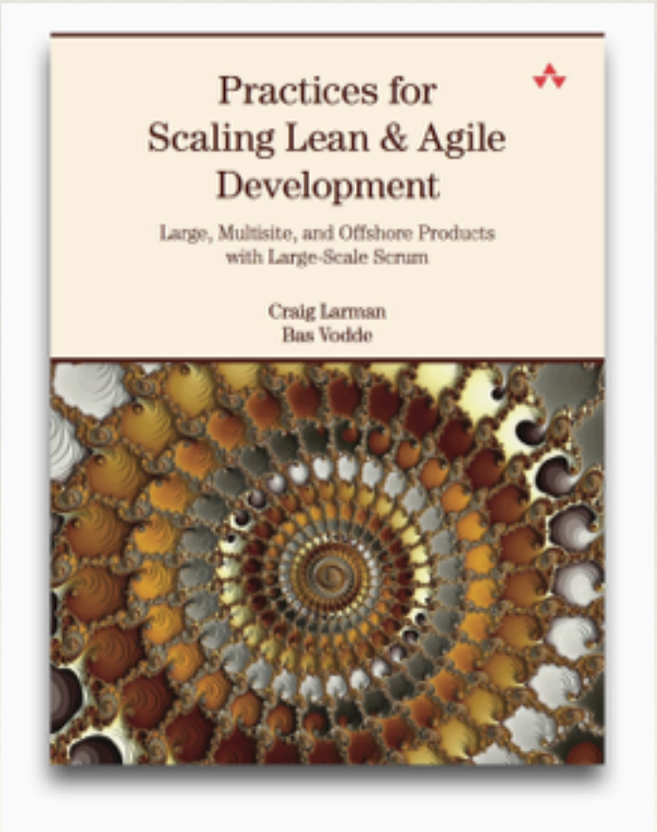
Who wrote the foreword for our book?

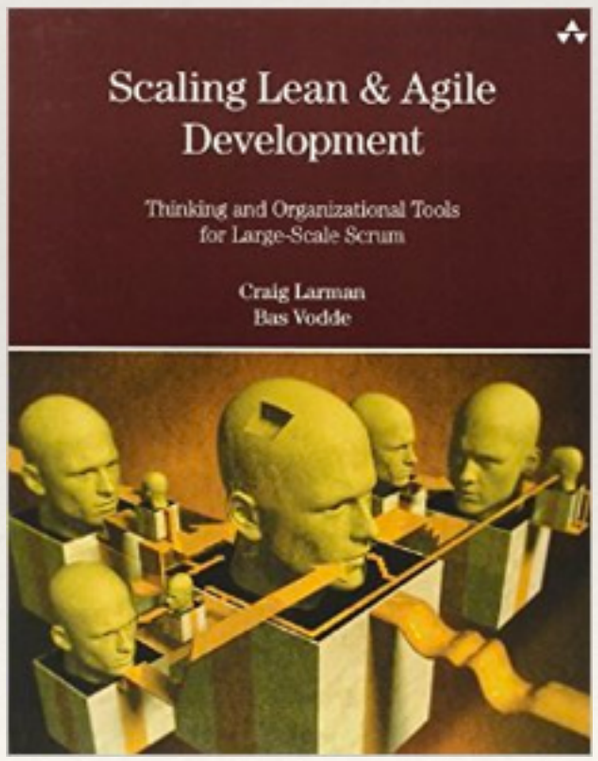




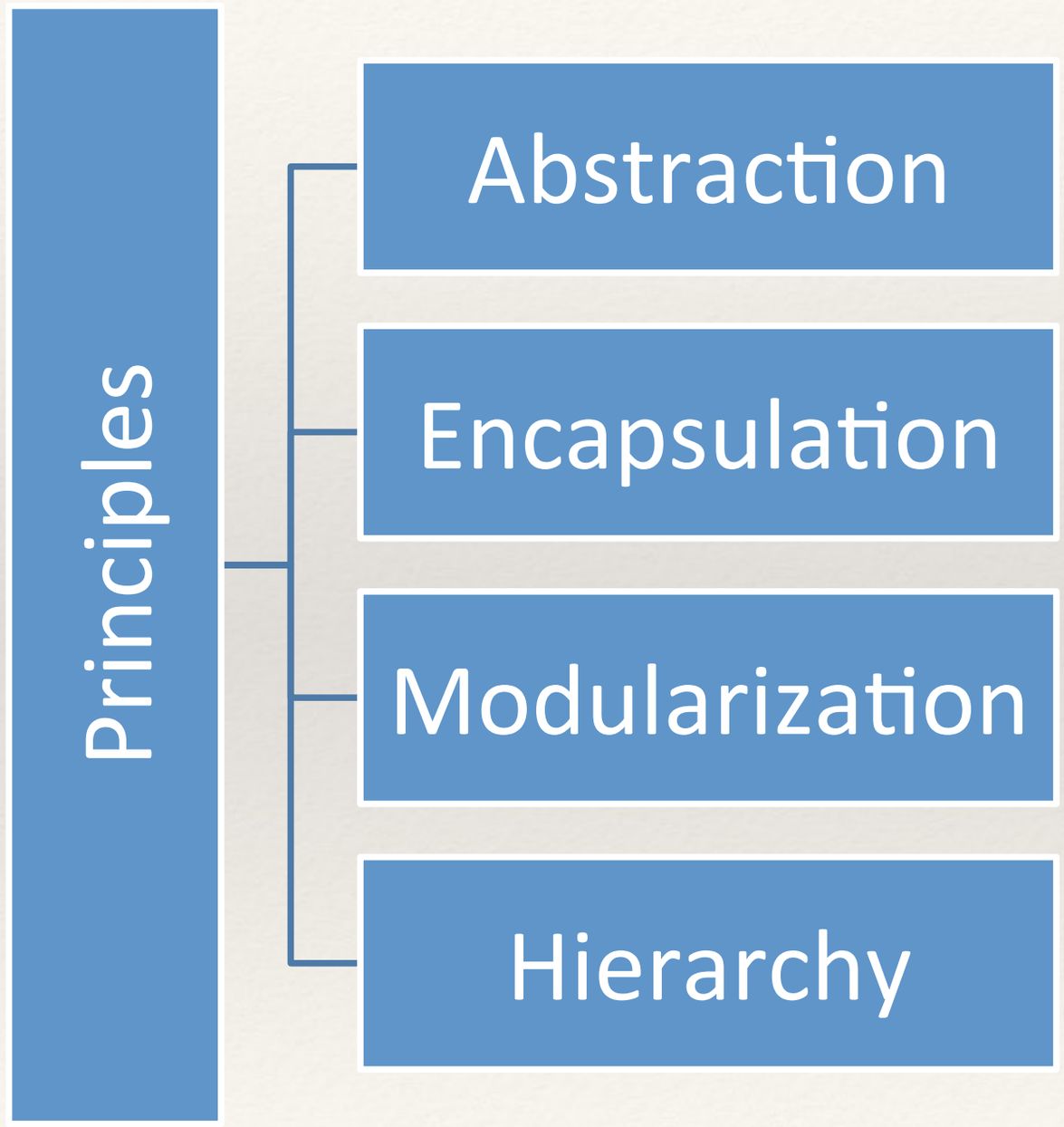
"The critical design tool for software development is a mind well educated in design principles"

–Craig Larman



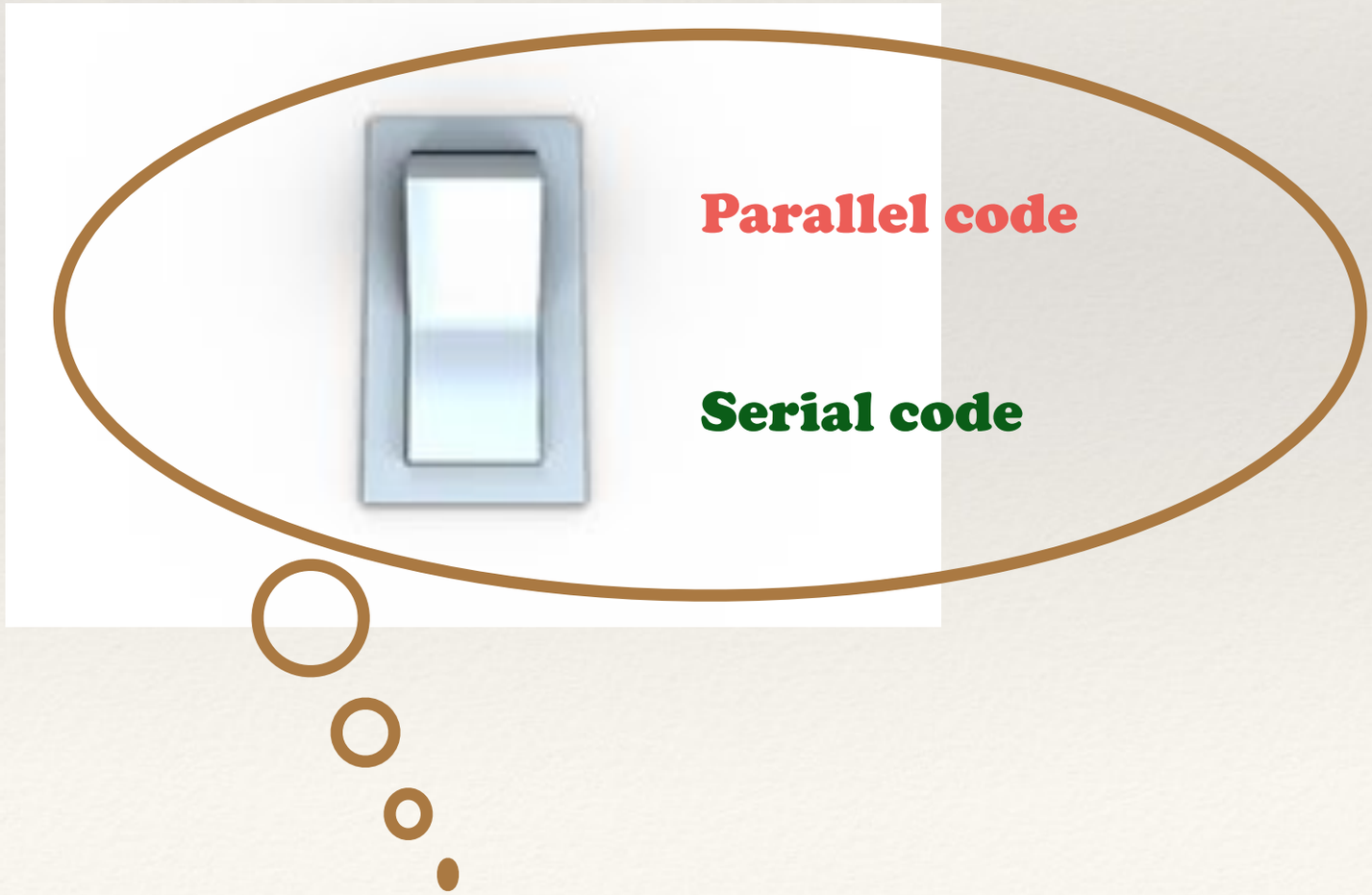


Fundamental principles in software design



Effective design: Java parallel streams

- Applies the principles of abstraction and encapsulation effectively to significantly simplify concurrent programming



Parallel streams: example

```
import java.util.stream.LongStream;

class PrimeNumbers {
   private static boolean isPrime(long val) {
      for(long i = 2; i <= val/2; i++) {
         if((val % i) == 0) {
            return false;
         }
      }
      return true;
   }
   public static void main(String []args) {
      long numOfPrimes = LongStream.rangeClosed(2, 50_000)
                .parallel()
                .filter(PrimeNumbers::isPrime)
                .count();
      System.out.println(numOfPrimes);
   }
}
```

Prints 9592

```
long numOfPrimes = LongStream.rangeClosed(2, 100_000)
                             .filter(PrimeNumbers::isPrime)
                             .count();
System.out.println(numOfPrimes);
```

Let’s flip the switch by calling parallel() function

Parallel code

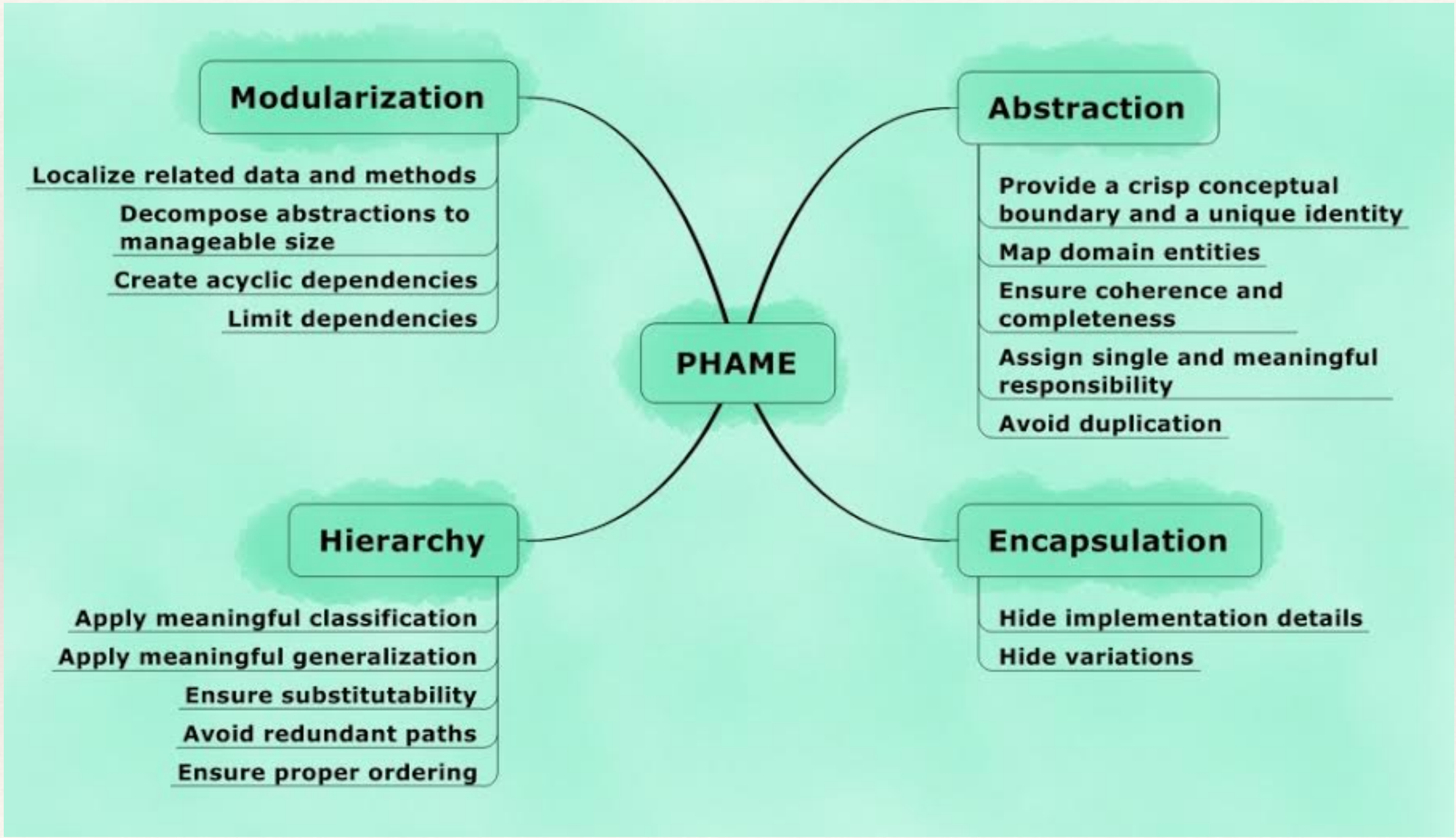
Serial code

Prints 9592

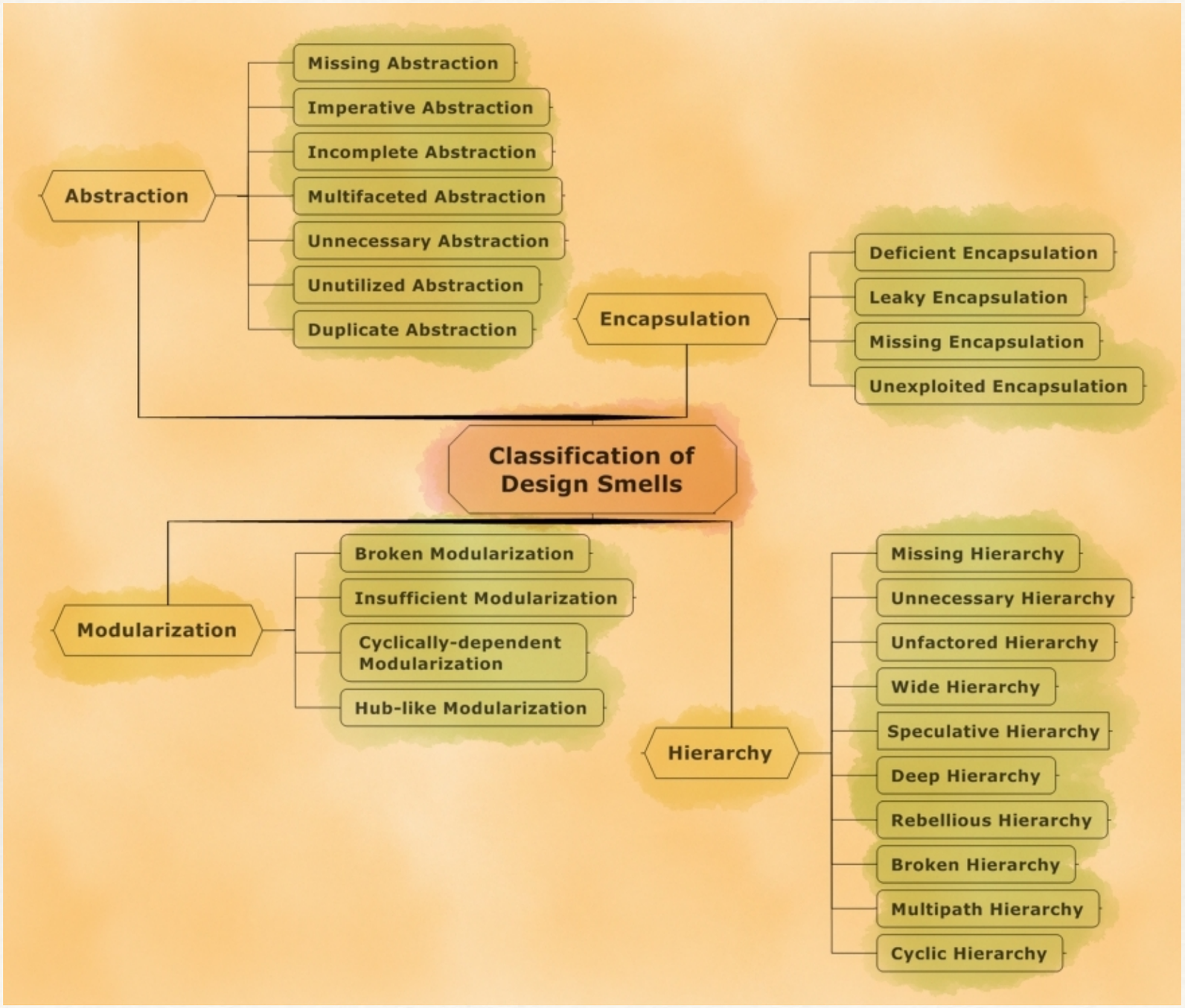
```
long numOfPrimes = LongStream.rangeClosed(2, 100_000)
                             .parallel()
                             .filter(PrimeNumbers::isPrime)
                             .count();
System.out.println(numOfPrimes);
```

1.235 seconds

Proactive application: enabling techniques



Reactive application: smells



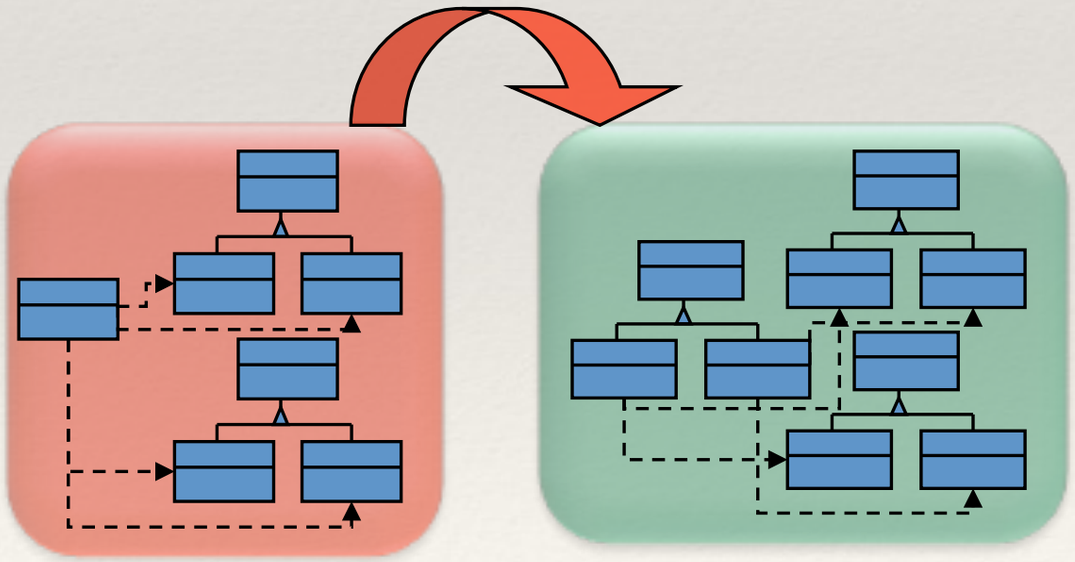
What are smells?

"Smells are certain structures
in the code that suggest
(sometimes they scream for)
the possibility of refactoring."

What is refactoring?

Refactoring (noun): a *change* made to the *internal structure* of software to make it *easier to understand and cheaper to modify* without *changing its observable behavior*

Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behavior



Should this be “refactored”?

```
abstract class Printer {
  private Integer portNumber = getPortNumber();
  abstract Integer getPortNumber();
  public static void main(String[]s) {
    Printer p = new LPDPrinter();
    System.out.println(p.portNumber);
  }
}

class LPDPrinter extends Printer {
  /* Line Printer Deamon port no is 515 */
  private Integer defaultPortNumber = 515;
  Integer getPortNumber() {
    return defaultPortNumber;
  }
}
```

Should this be “refactored”?

Does this have “long message chains” smell?

```
Calendar calendar = new Calendar.Builder()
.set(YEAR, 2003)
.set(MONTH, APRIL)
.set(DATE, 6)
.set(HOUR, 15)
.set(MINUTE, 45)
.set(SECOND, 22)
.setTimeZone(TimeZone.getDefault())
.build();
System.out.println(calendar);
```

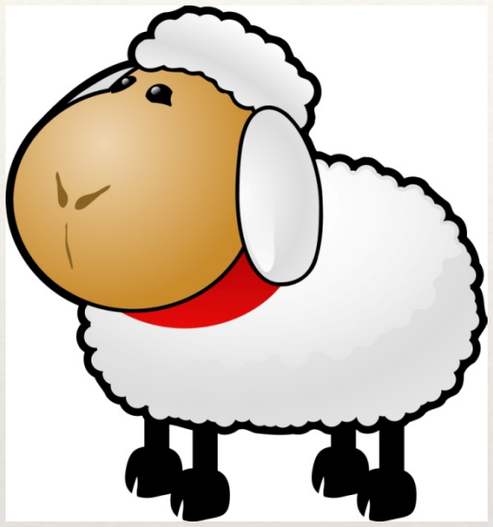
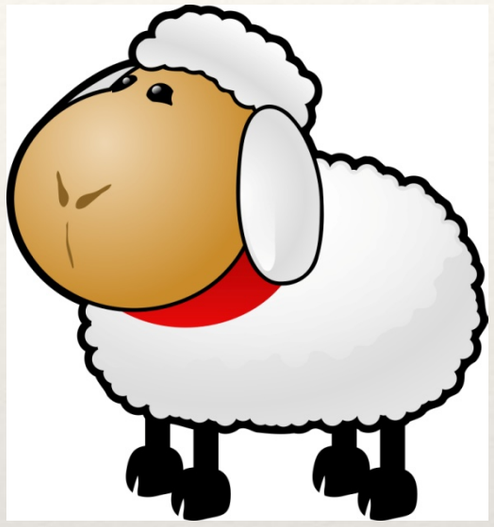
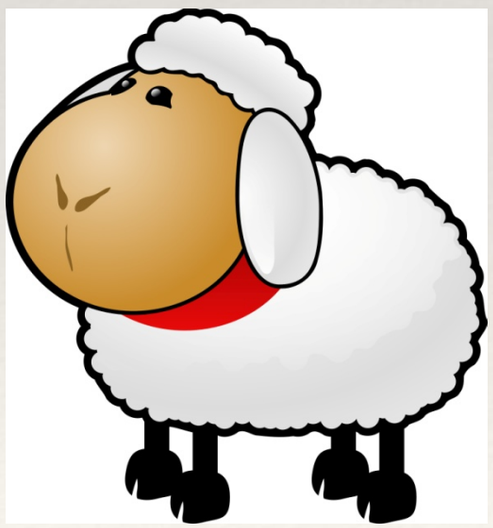
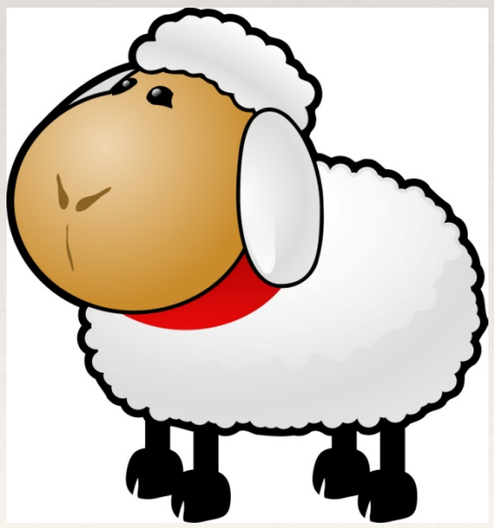
No - it’s a feature
(use of builder pattern)!

Design smells: Example

"Lazy class" smell

```
public class FormattableFlags {
        // Explicit instantiation of this class is prohibited.
        private FormattableFlags() {}
        /** Left-justifies the output.  */
        public static final int LEFT_JUSTIFY = 1<<0; // '-'
        /** Converts the output to upper case */
        public static final int UPPERCASE = 1<<1;    // 'S'
        /**Requires the output to use an alternate form. */
        public static final int ALTERNATE = 1<<2;    // '#'
}
```

Duplicated Code

Duplicated Code

Type 1

- **exactly identical** except for variations in whitespace, layout, and comments

Type 2

- **syntactically identical** except for variation in symbol names, whitespace, layout, and comments

Type 3

- identical except some **statements changed, added, or removed**

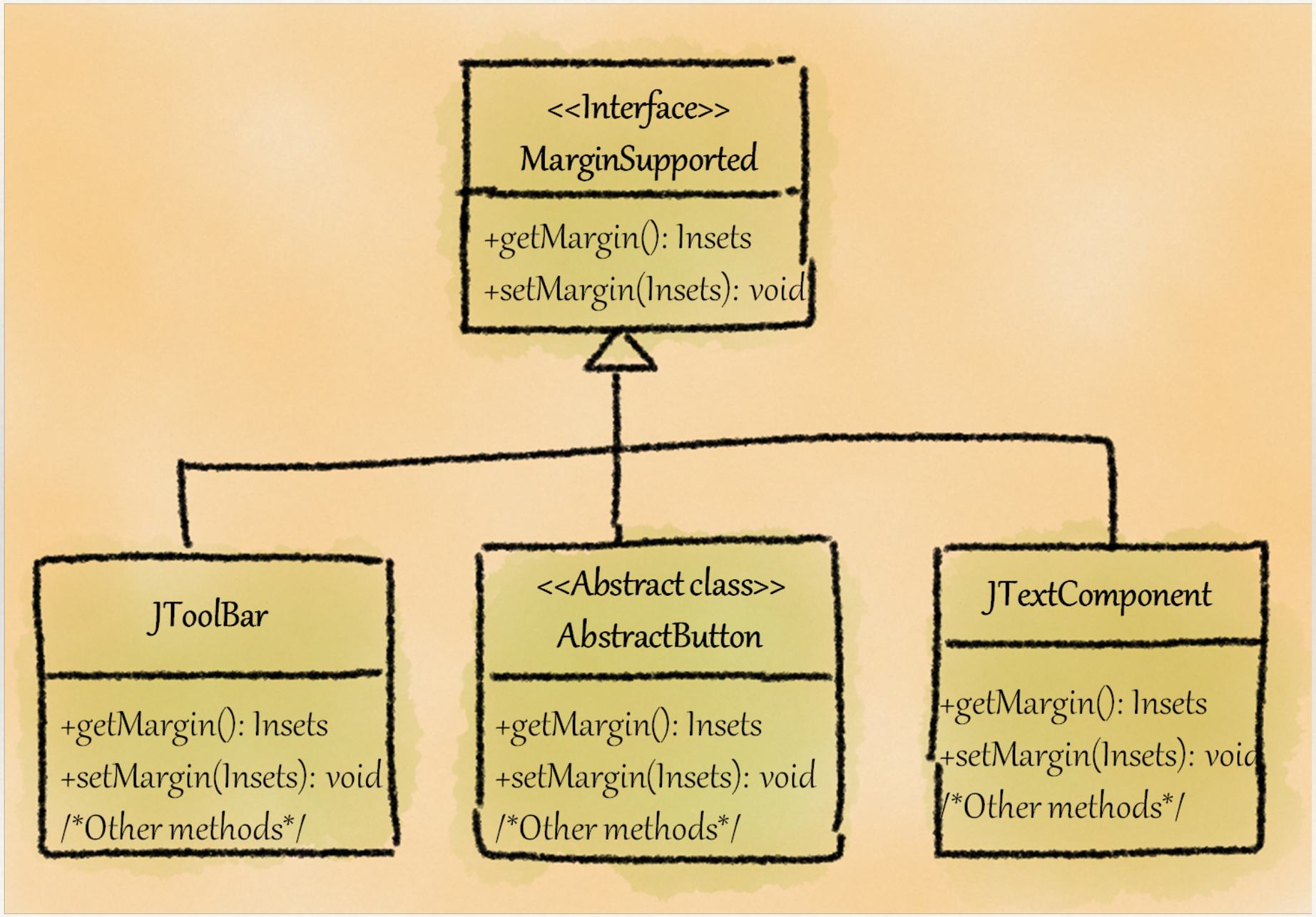
Type 4

- when the fragments are **semantically identical** but implemented by syntactic variants

What's that smell?

```
public Insets getBorderInsets(Component c, Insets insets){
        Insets margin = null;
         // Ideally we'd have an interface defined for classes which
        // support margins (to avoid this hackery), but we've
        // decided against it for simplicity
        //
        if (c instanceof AbstractButton) {
                margin = ((AbstractButton)c).getMargin();
        } else if (c instanceof JToolBar) {
                margin = ((JToolBar)c).getMargin();
        } else if (c instanceof JTextComponent) {
                margin = ((JTextComponent)c).getMargin();
        }
        // rest of the code omitted …
```

Refactoring

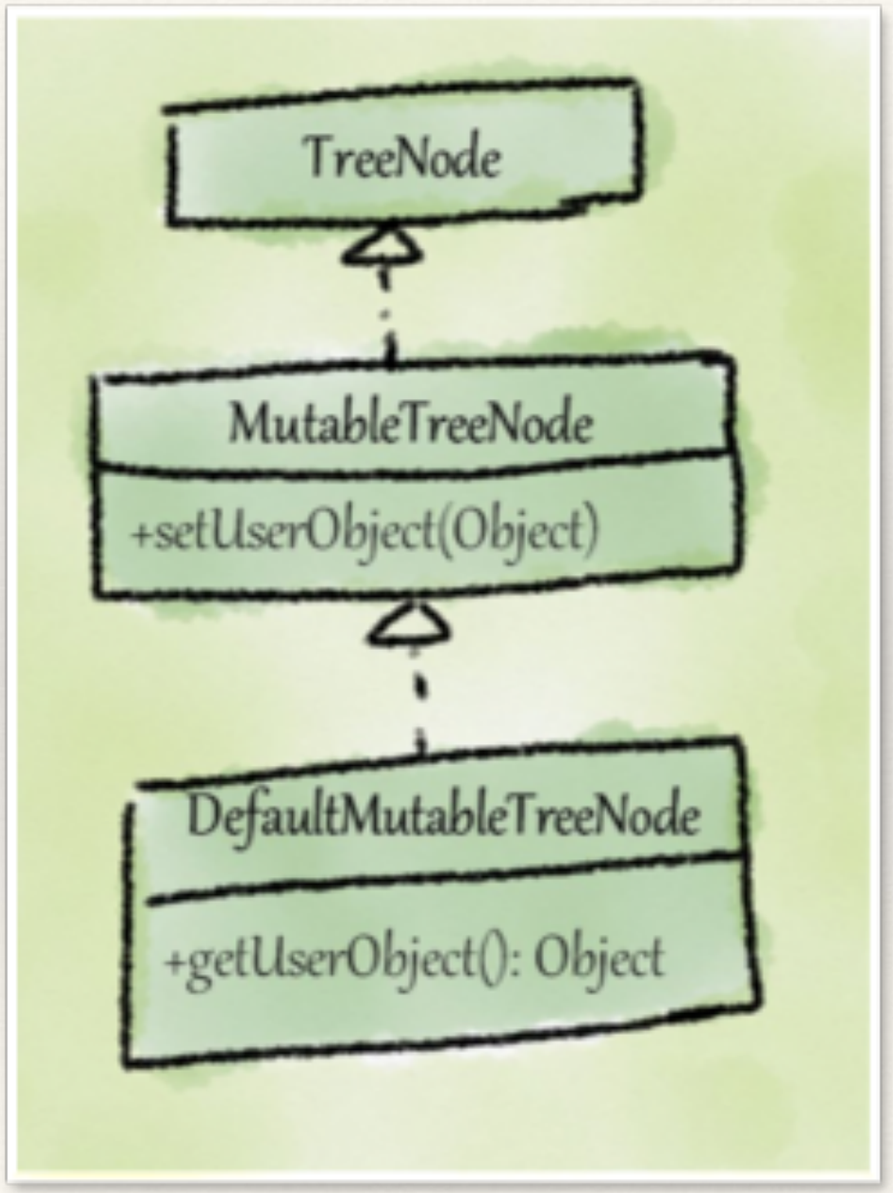


Refactoring

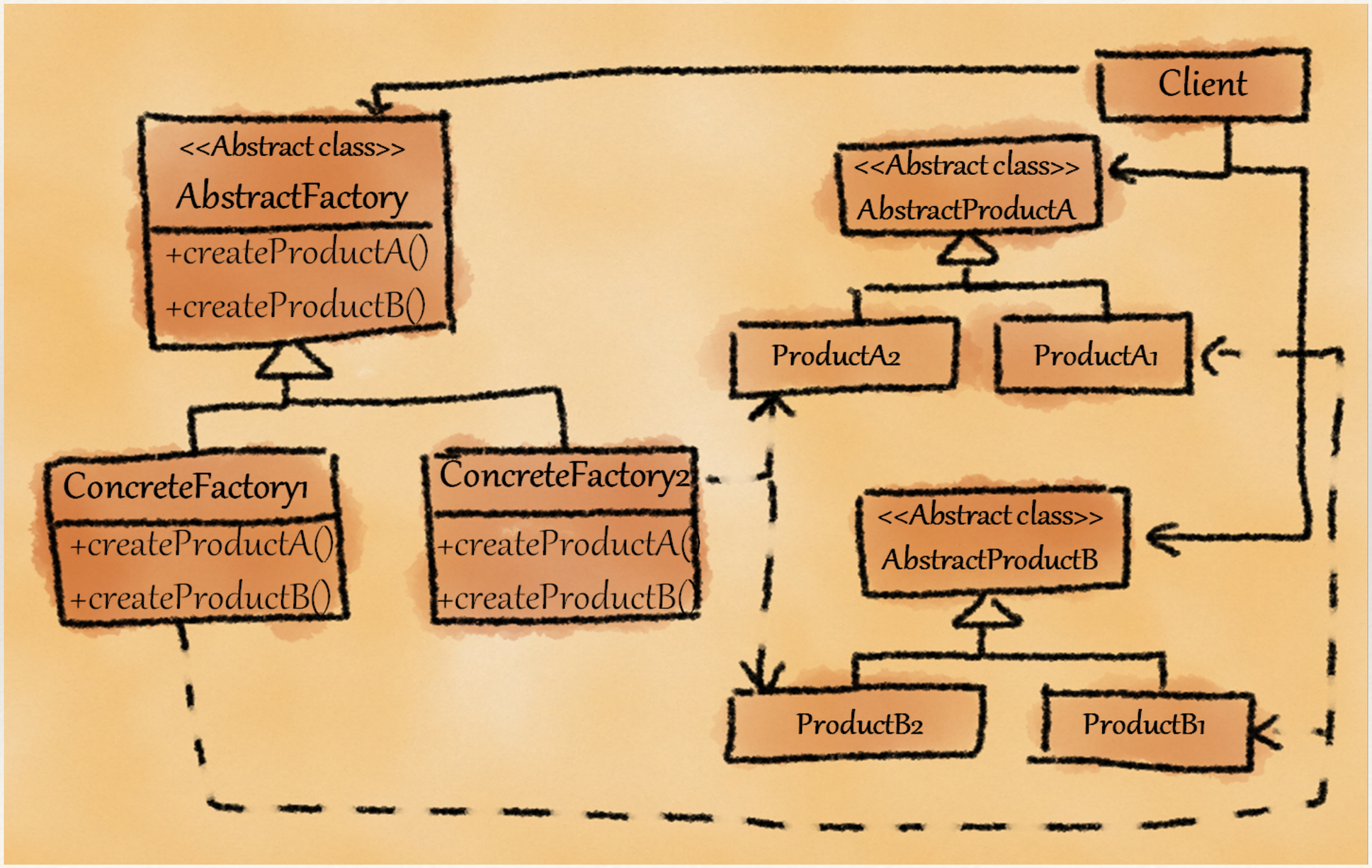
```
if (c instanceof AbstractButton) {
        margin = ((AbstractButton)c).getMargin();
} else if (c instanceof JToolBar) {
        margin = ((JToolBar)c).getMargin();
} else if (c instanceof JTextComponent) {
        margin = ((JTextComponent)c).getMargin();
}
```

```
margin = c.getMargin();
```

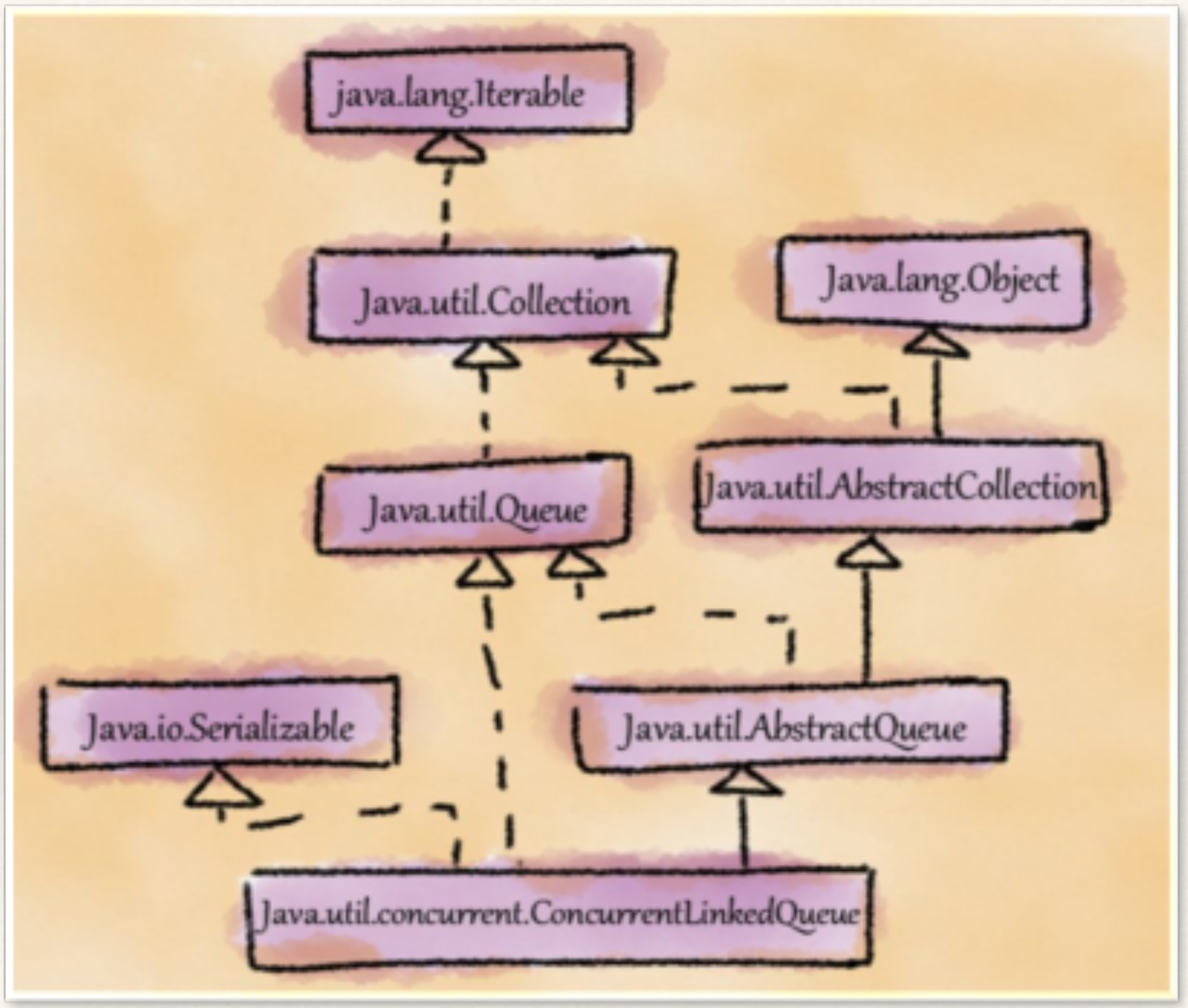
Design smells: Example



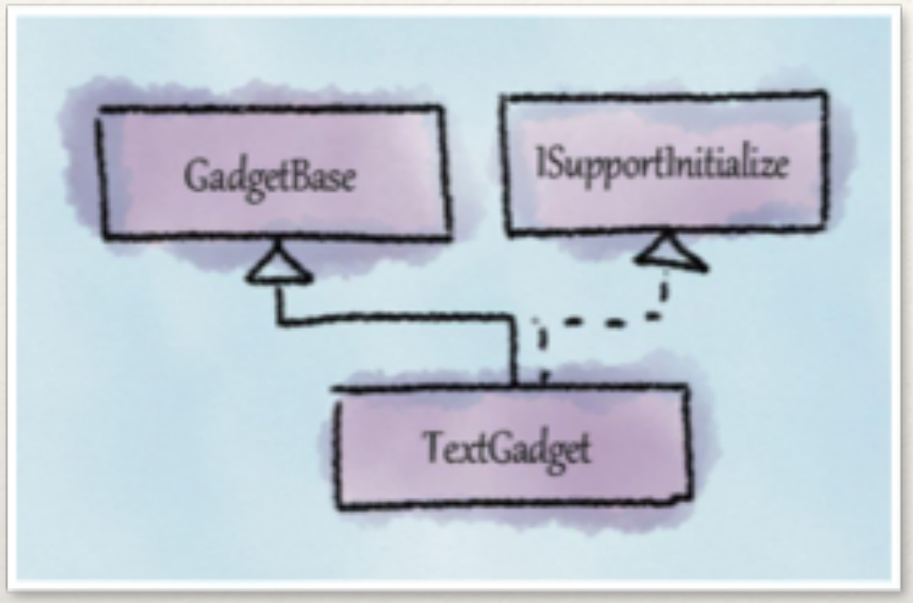
Discussion example

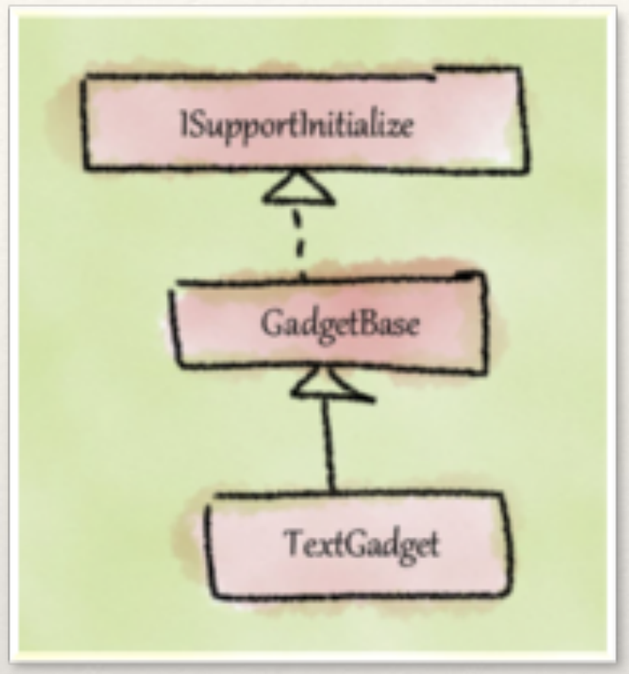


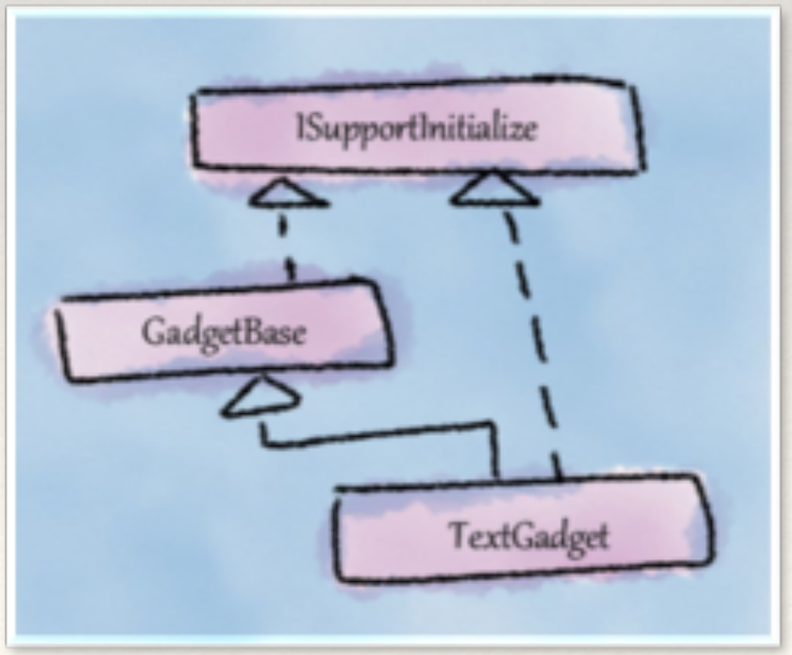
Design smells: Example



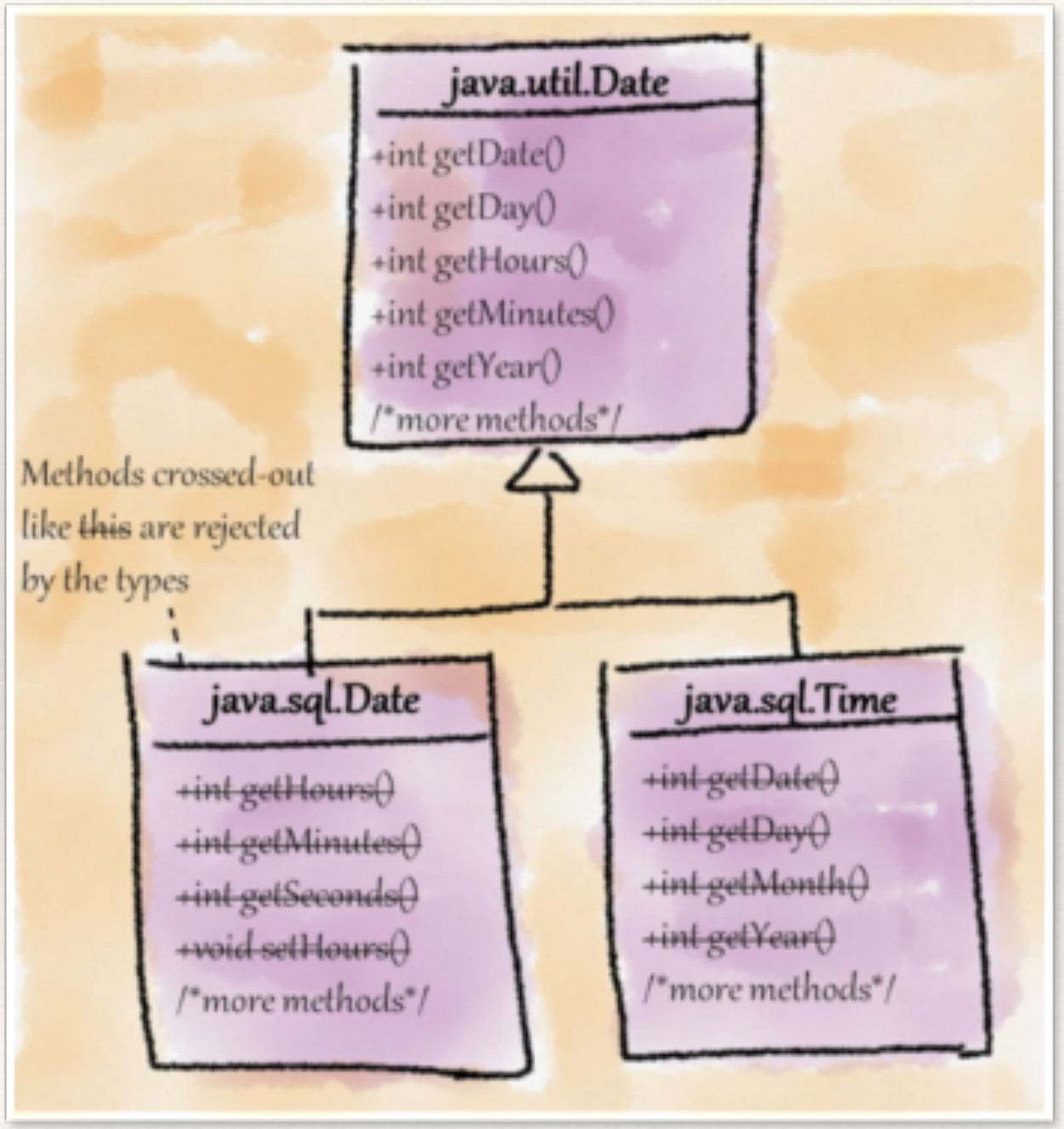
Discussion example







Design smells: example #3



Discussion example

```
// using java.util.Date
Date today = new Date();
System.out.println(today);
```

```
$ java DateUse
Wed Dec 02 17:17:08 IST 2015
```

Why should we get the time and timezone details if I only want a date? Can I get rid of these parts? No!

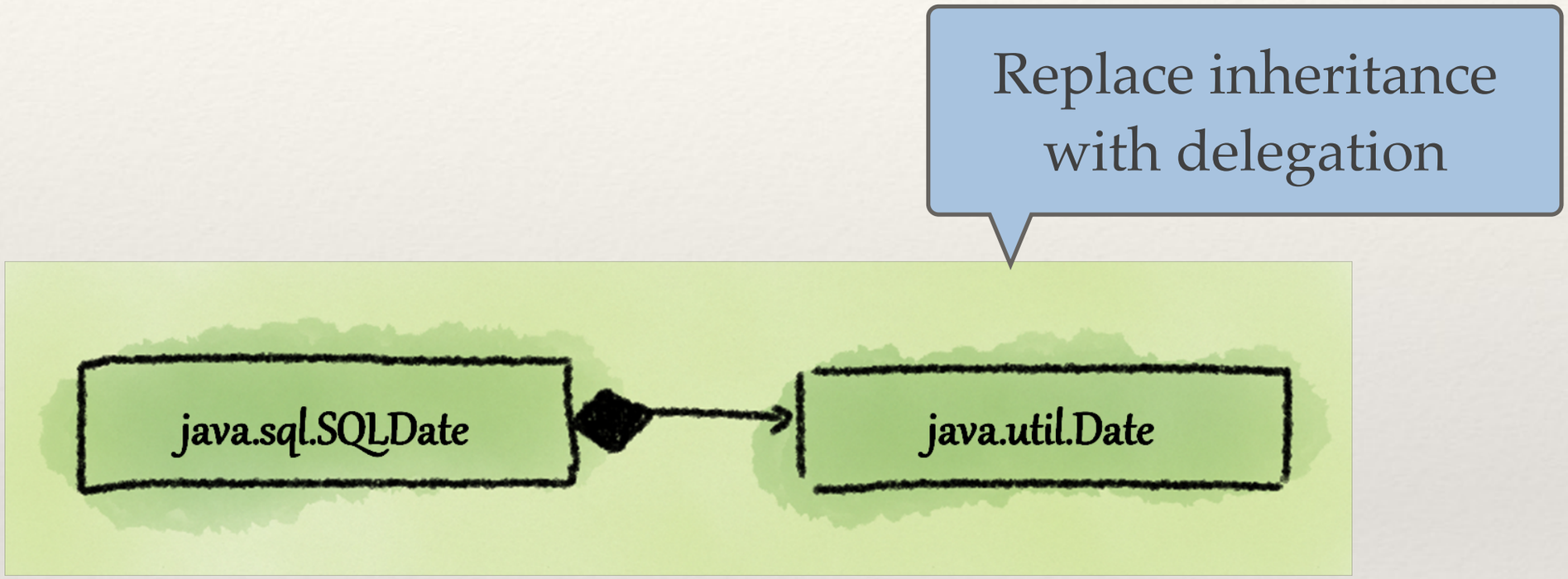
So what!

```
Date today = new Date();
System.out.println(today);
Date todayAgain = new Date();
System.out.println(todayAgain);
System.out.println(today.compareTo(todayAgain) == 0);
```

```
Thu Mar 17 13:21:55 IST 2016
Thu Mar 17 13:21:55 IST 2016
false
```

What is going on here?

Refactoring for Date



Joda API

Stephen Colebourne

java.time package!

Date, Calendar, and TimeZone

Java 8 replaces these types

Refactored solution

```
LocalDate today = LocalDate.now();
System.out.println(today);
LocalDate todayAgain = LocalDate.now();
System.out.println(todayAgain);
System.out.println(today.compareTo(todayAgain) == 0);
```

```
2016-03-17
2016-03-17
true
```

Works fine now!

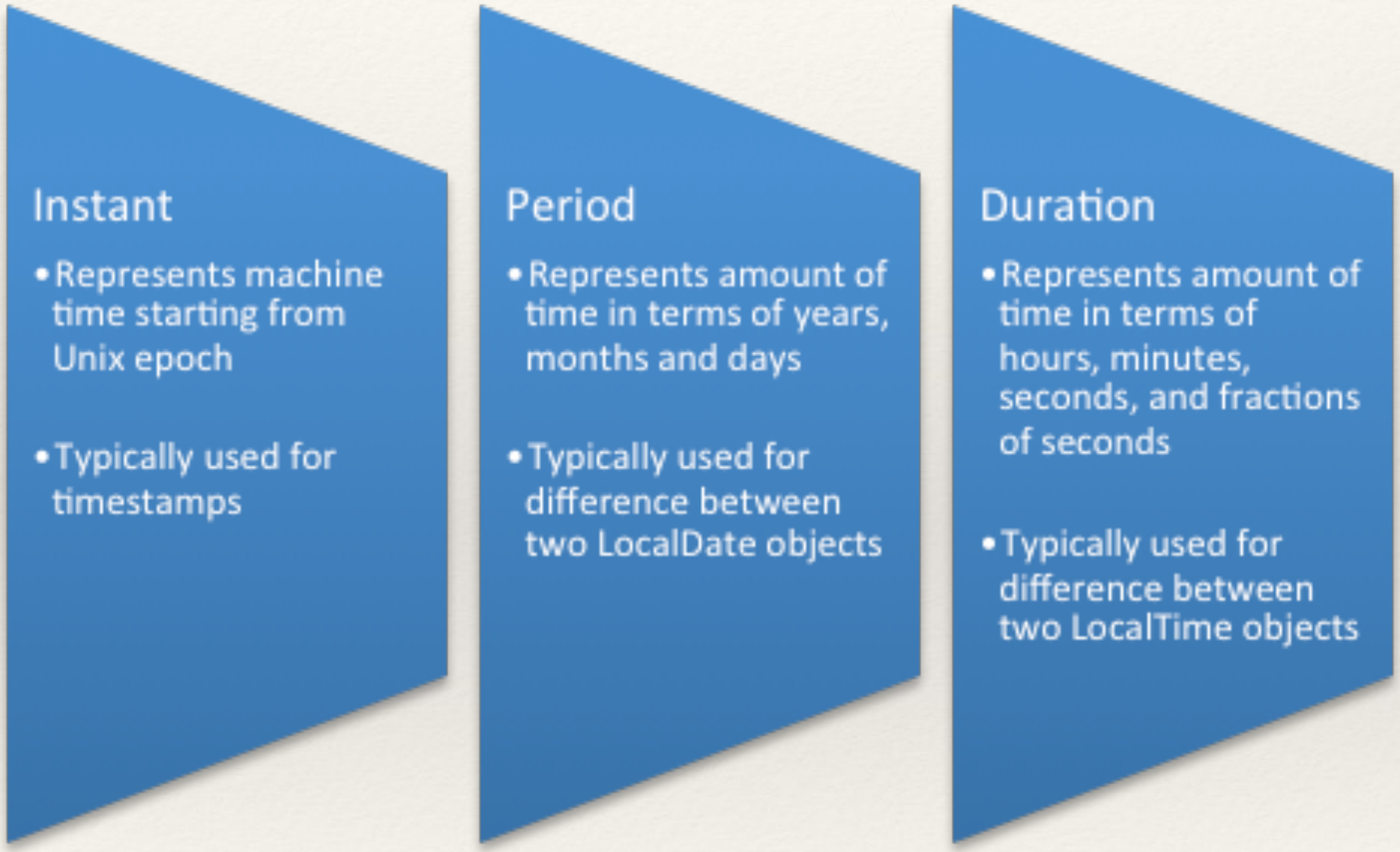
Refactored example ...

```
LocalDate today = LocalDate.now();
System.out.println(today);

LocalTime now = LocalTime.now();
System.out.println(now);

ZoneId id = ZoneId.of("Asia/Tokyo");
System.out.println(id);

LocalDateTime todayAndNow = LocalDateTime.now();
System.out.println(todayAndNow);

ZonedDateTime todayAndNowInTokyo = ZonedDateTime.now(ZoneId.of("Asia/Tokyo"));
System.out.println(todayAndNowInTokyo);
```

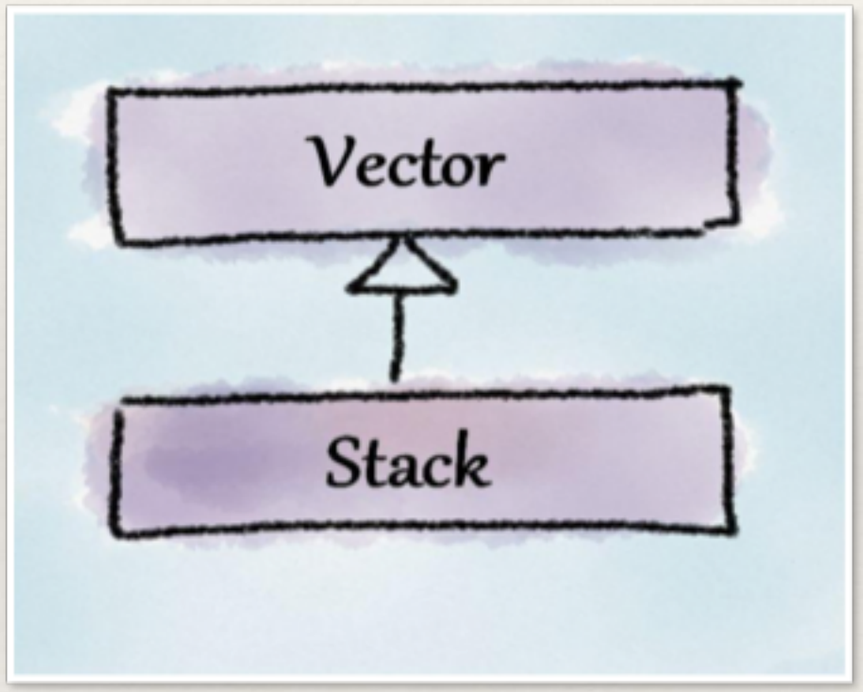
You can use only date, time, or even timezone, and combine them as needed!

```
2016-03-17
13:28:06.927
Asia/Tokyo
2016-03-17T13:28:06.928
2016-03-17T16:58:06.929+09:00[Asia/Tokyo]
```

More classes in Date/Time API



What's that smell?



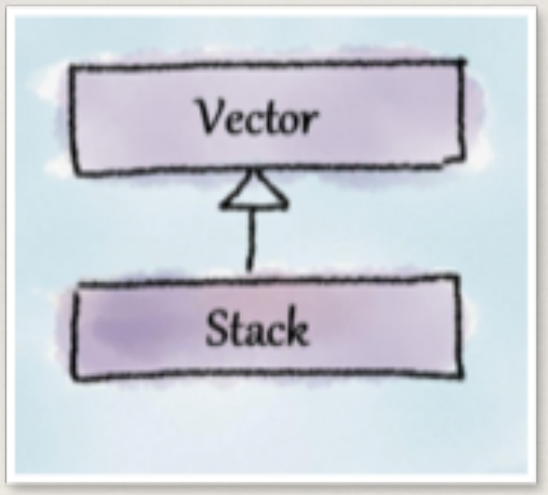
Liskov's Substitution Principle (LSP)

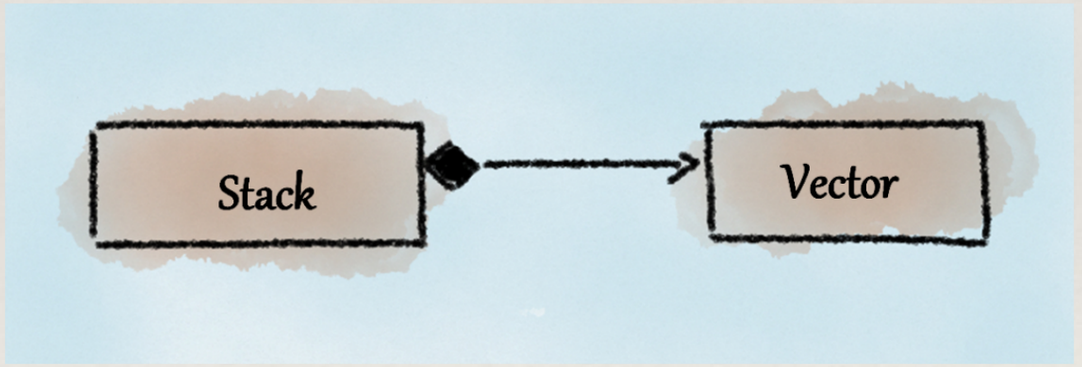
It should be possible to replace objects of supertype with objects of subtypes without altering the desired behavior of the program

Barbara Liskov

Refactoring

Replace inheritance with delegation



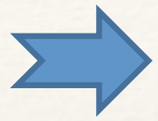


What's that smell?

```
switch (transferType) {
        case DataBuffer.TYPE_BYTE:
                byte bdata[] = (byte[])inData;
                pixel = bdata[0] & 0xff;
                length = bdata.length;
                break;
        case DataBuffer.TYPE_USHORT:
                short sdata[] = (short[])inData;
                pixel = sdata[0] & 0xffff;
                length = sdata.length;
                break;
        case DataBuffer.TYPE_INT:
                int idata[] = (int[])inData;
                pixel = idata[0];
                length = idata.length;
                break;
        default:
                throw new UnsupportedOperationException("This method has not been "+ "implemented
                for transferType " + transferType);
}
```

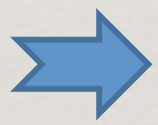
Replace conditional with polymorphism

protected int transferType;



protected DataBuffer dataBuffer;

```
switch (transferType) {
        case DataBuffer.TYPE_BYTE:
                byte bdata[] = (byte[])inData;
                pixel = bdata[0] & 0xff;
                length = bdata.length;
                break;
        case DataBuffer.TYPE_USHORT:
                short sdata[] = (short[])inData;
                pixel = sdata[0] & 0xffff;
                length = sdata.length;
                break;
        case DataBuffer.TYPE_INT:
                int idata[] = (int[])inData;
                pixel = idata[0];
                length = idata.length;
                break;
        default:
                throw new UnsupportedOperationException("This method
                has not been "+ "implemented for transferType " +
                transferType);
}
```



```
pixel = dataBuffer.getPixel();
length = dataBuffer.getSize();
```

Scenario

```
class Plus extends Expr {
        private Expr left, right;
        public Plus(Expr arg1, Expr arg2) {
                left = arg1;
                right = arg2;
        }
        public void genCode() {
                left.genCode();
                right.genCode();
                if(t == Target.JVM) {
                        System.out.println("iadd");
                }
                else { / / DOTNET
                        System.out.println("add");
                }
        }
}
```

How to separate:

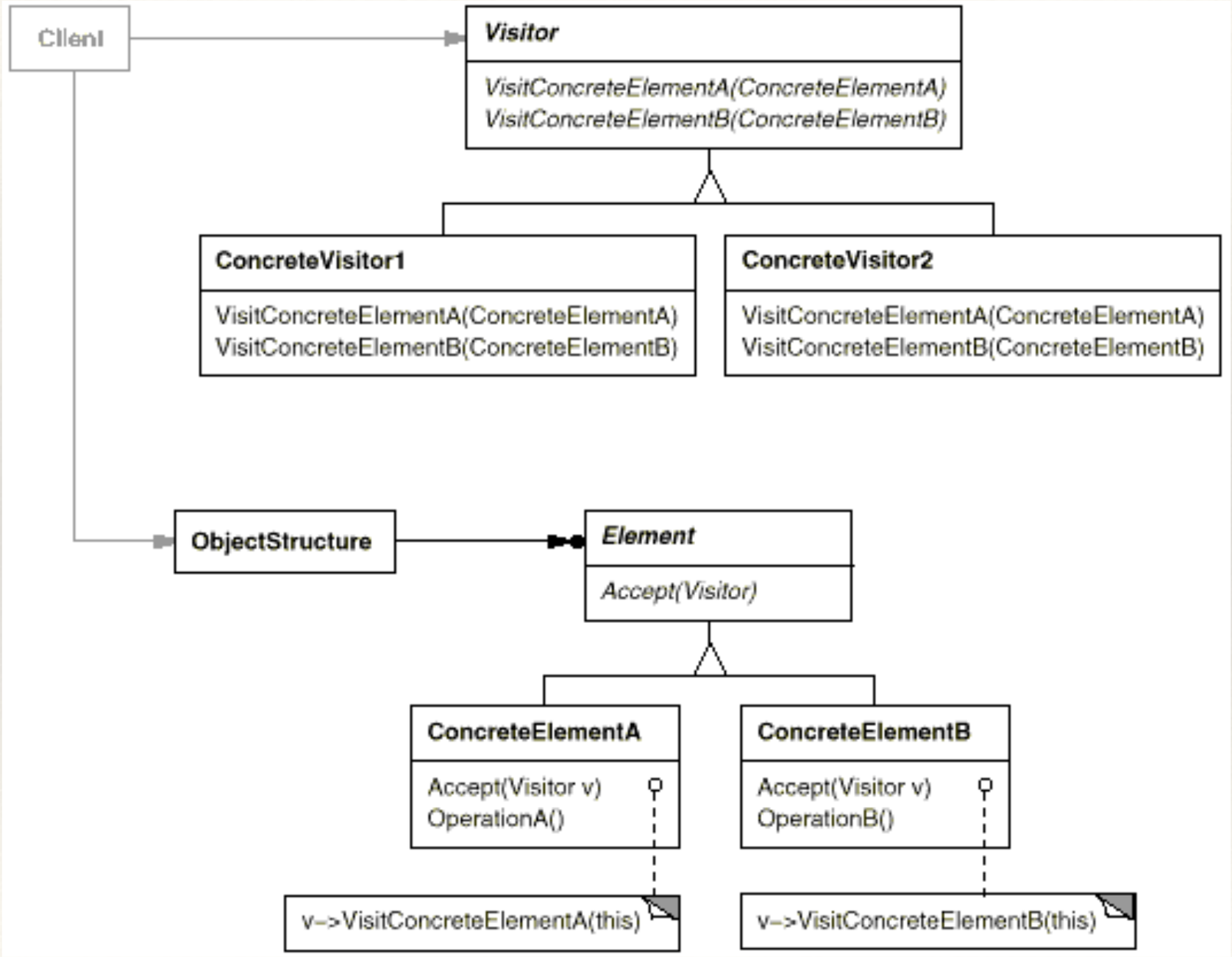
a) code generation logic from node types?
b) how to support different target types?

A solution using Visitor pattern

```
class Plus extends Expr {
        private Expr left, right;
        public Plus(Expr arg1, Expr arg2) {
                left = arg1;
                right = arg2;
        }
        public Expr getLeft() {
                return left;
        }
        public Expr getRight() {
                return right;
        }
        public void accept(Visitor v) {
                v.visit(this);
        }
}
```

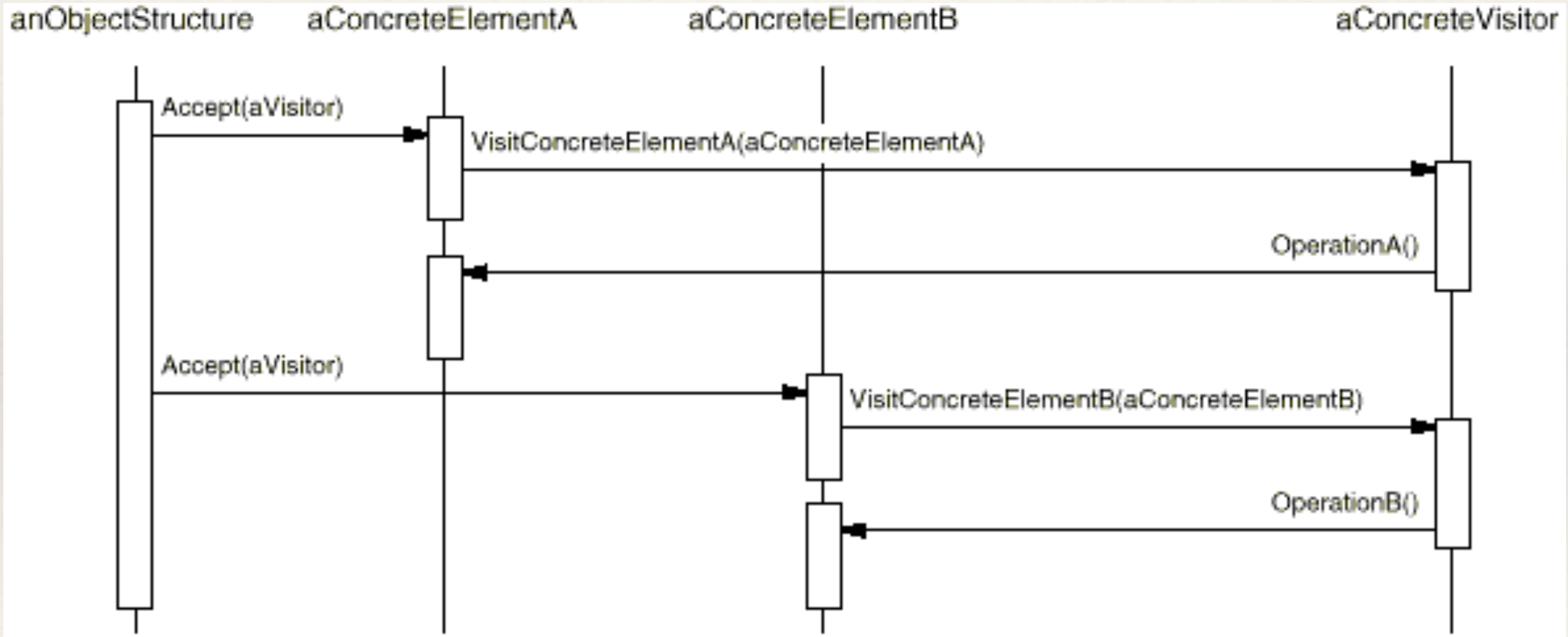
```
class DOTNETVisitor extends Visitor {
        public void visit(Constant arg) {
                System.out.println("ldarg " + arg.getVal());
        }
        public void visit(Plus plus) {
                genCode(plus.getLeft());
                genCode(plus.getRight());
                System.out.println("add");
        }
        public void visit(Sub sub) {
                genCode(sub.getLeft());
                genCode(sub.getRight());
                System.out.println("sub");
        }
        public void genCode(Expr expr) {
                expr.accept(this);
        }
}
```

Visitor pattern: structure



Source: “Design Patterns: Elements of Reusable Object-Oriented Software”, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley,1994

Visitor pattern: call sequence



Source: “Design Patterns: Elements of Reusable Object-Oriented Software”, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley,1994

Visitor pattern: Discussion

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operations

- Create two class hierarchies:
  - One for the elements being operated on
  - One for the visitors that define operations on the elements

Refactoring with Lambdas

```
File[] files = new File(".").listFiles(
     new FileFilter() {
          public boolean accept(File f) { return f.isFile(); }
     }
);
for(File file: files) {
     System.out.println(file);
}
```

```
Arrays.stream(new File(“.”)
      .listFiles(file -> file.isFile()))
      .forEach(System.out::println);
```

Refactoring APIs

```
public class Throwable {
        // following method is available from Java 1.0 version.
        // Prints the stack trace as a string to standard output
        // for processing a stack trace,
        // we need to write regular expressions
        public void printStackTrace();
        // other methods omitted
}
```

Refactoring: Practical concerns

"Fear of breaking working code"

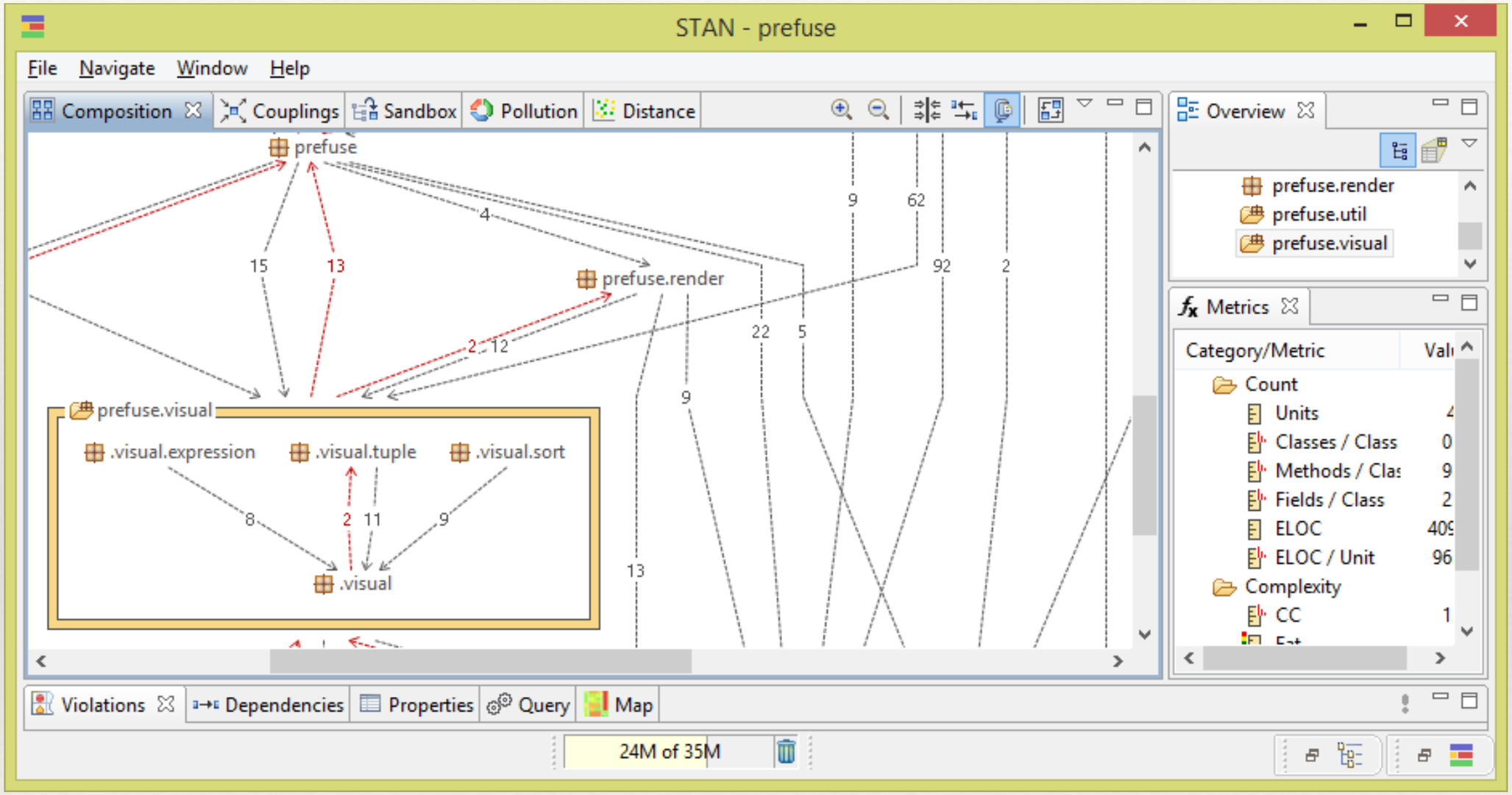
Where do I have time for refactoring?

How to refactor code in legacy projects (no automated tests, difficulty in understanding, lack of motivation, …)?

Is management buy-in necessary for refactoring?

Tool driven approach for design quality

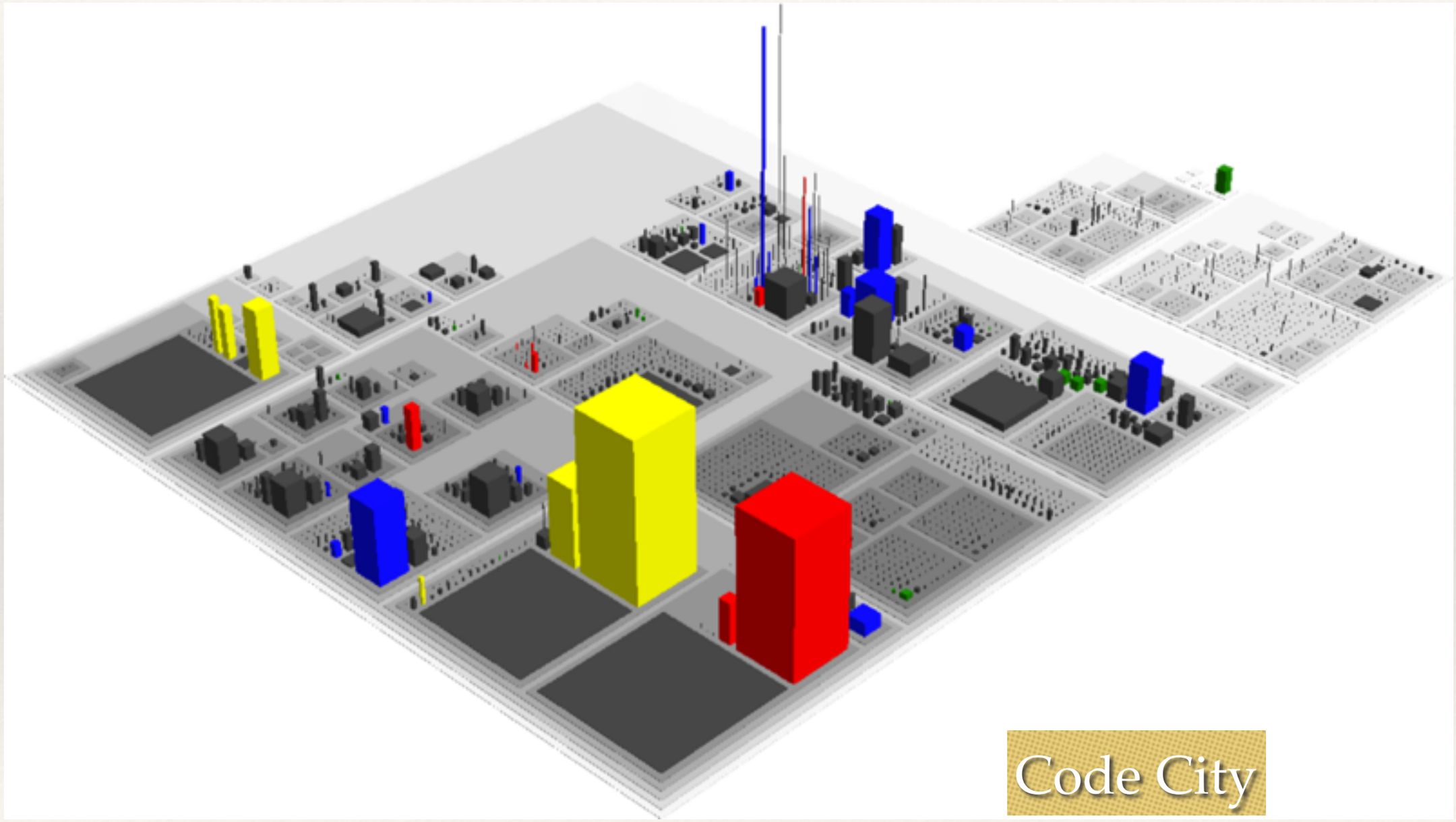
Comprehension tools



STAN

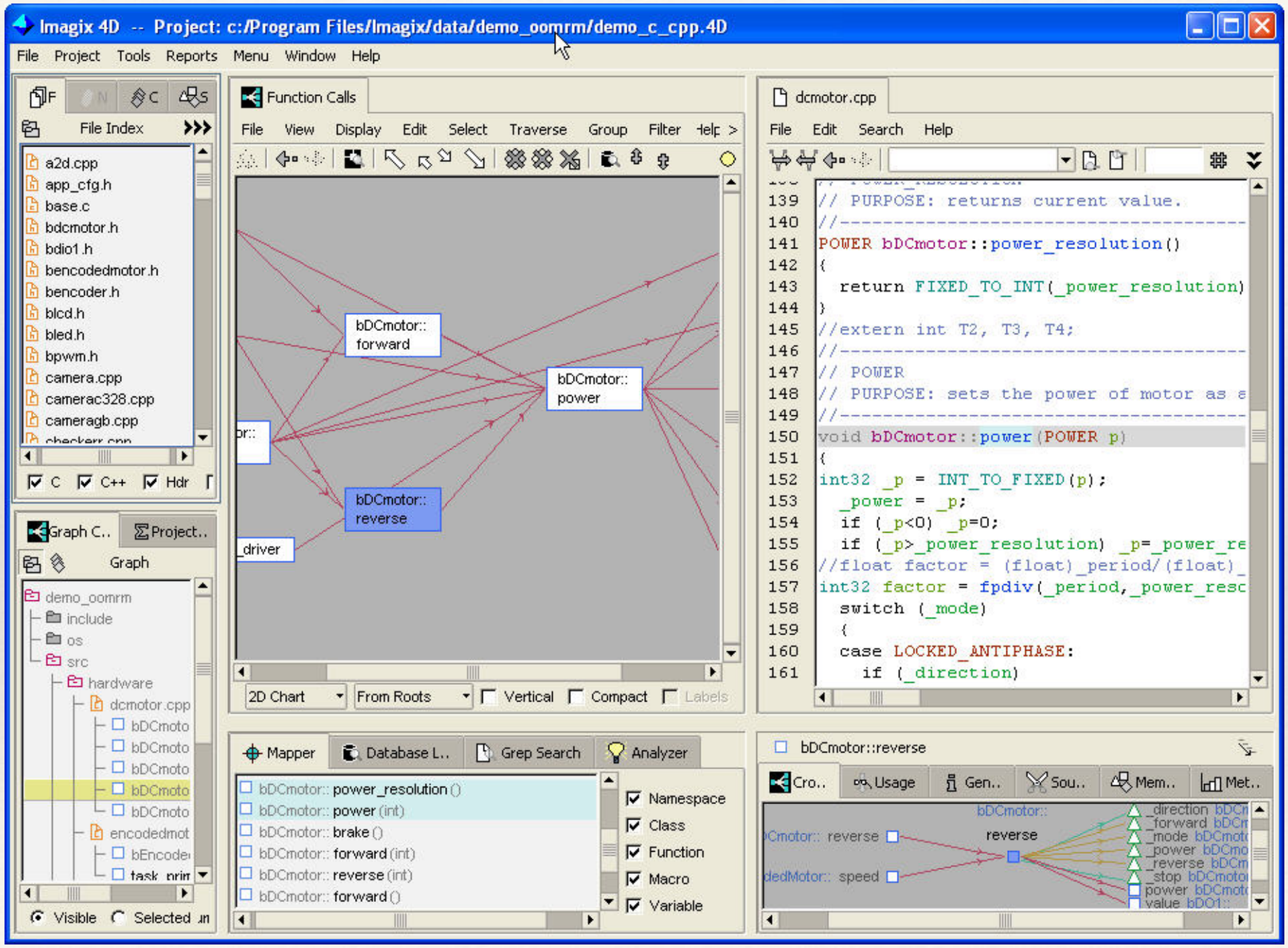
http://stan4j.com

Comprehension tools



http://www.inf.usi.ch/phd/wettel/codecity.html

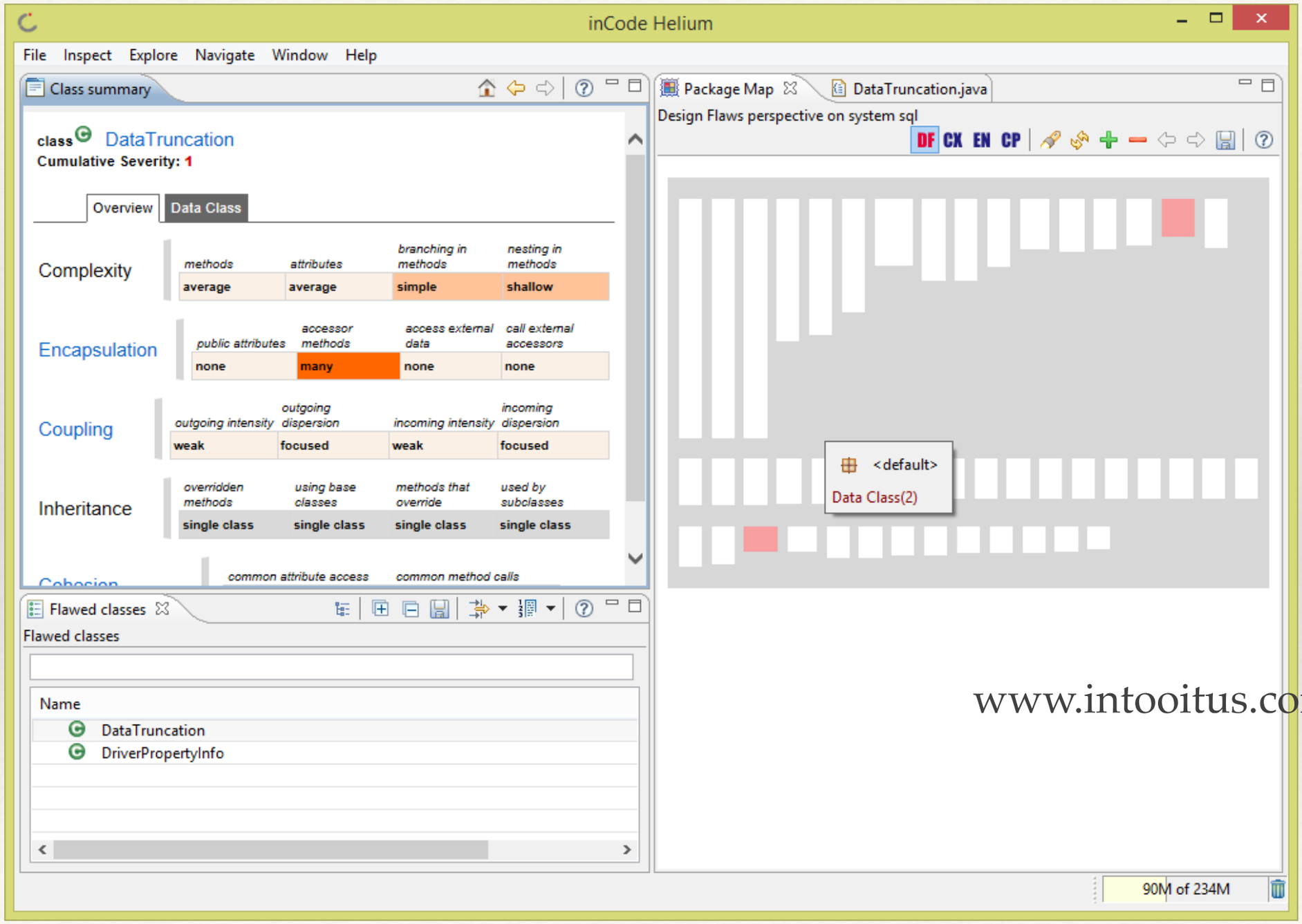
Comprehension tools



Imagix 4D

http://www.imagix.com

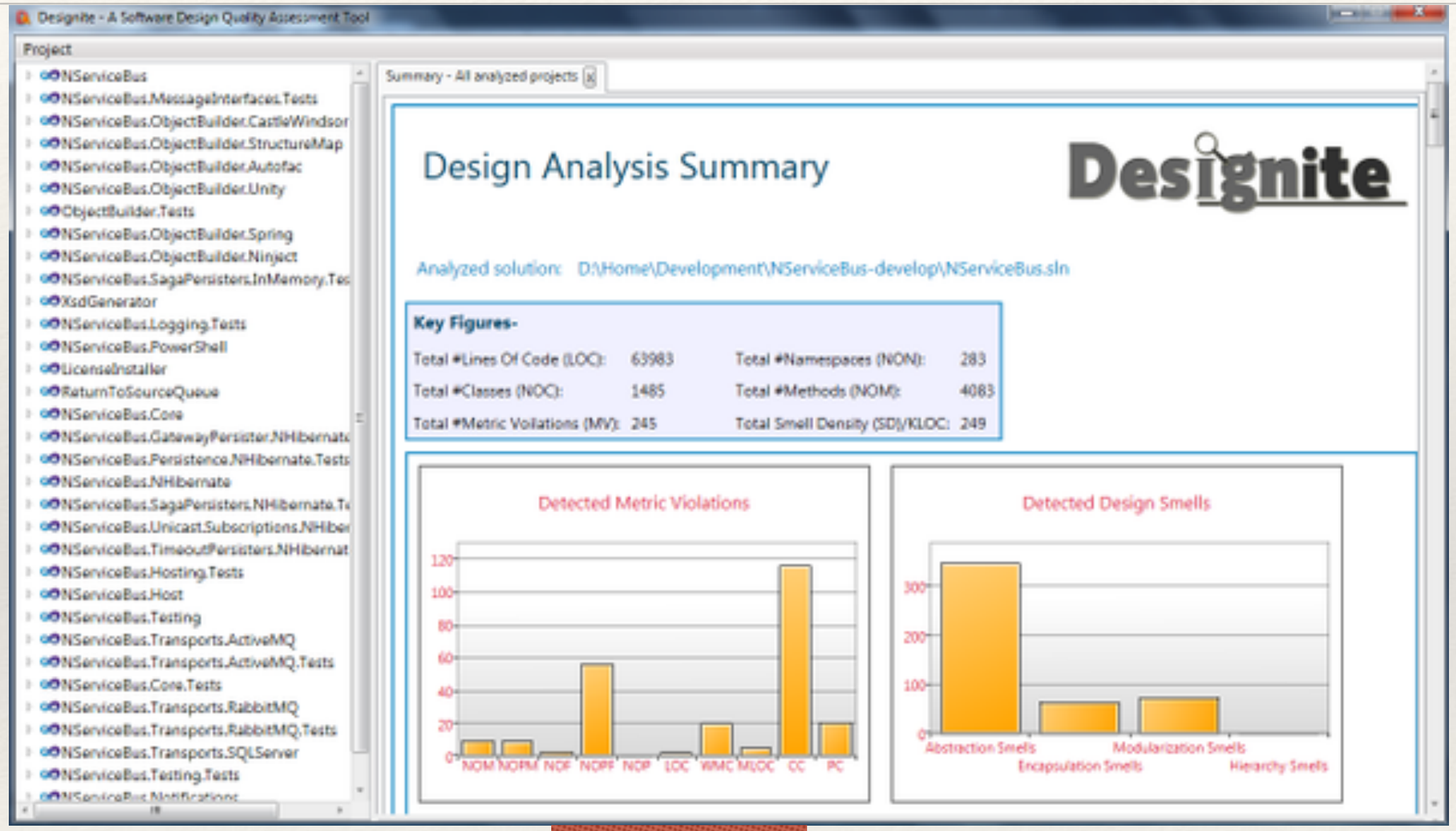
Smell detection tools



Infusion

www.intooitus.com/products/infusion

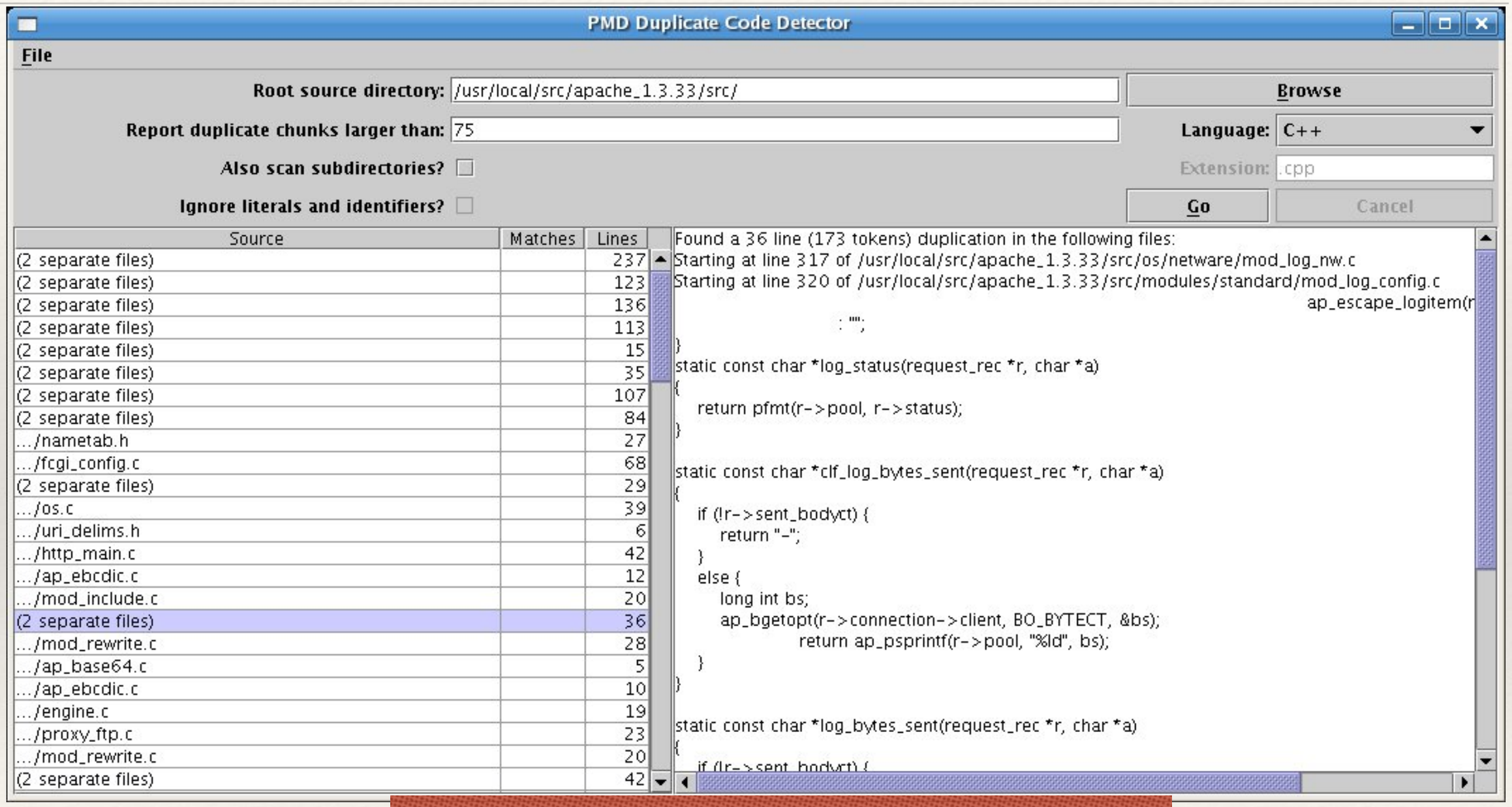
Smell detection tools



Designite

www.designite-tools.com

Clone analysers



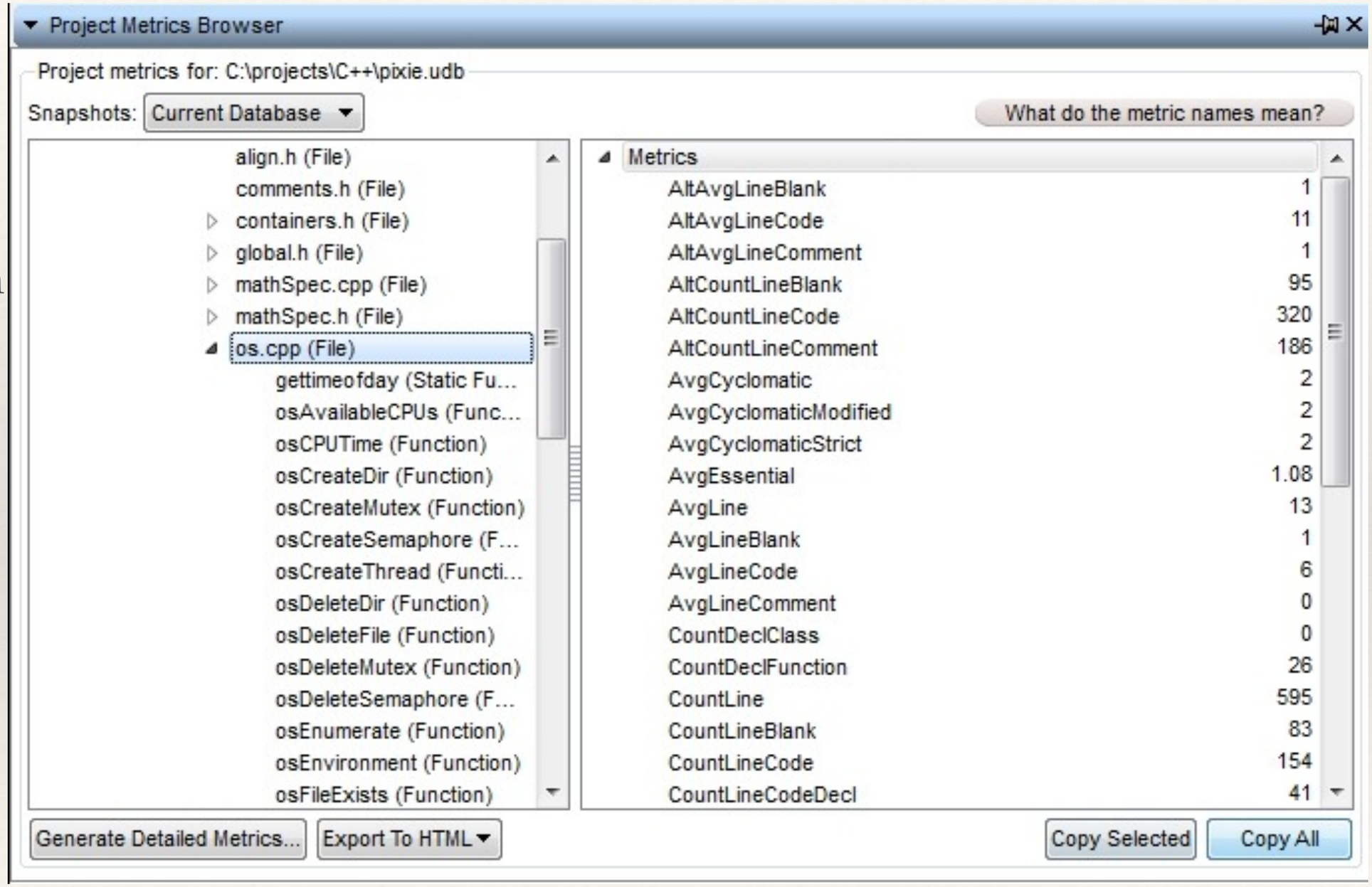
PMD Copy Paste Detector (CPD)

http://pmd.sourceforge.net/pmd-4.3.0/cpd.html

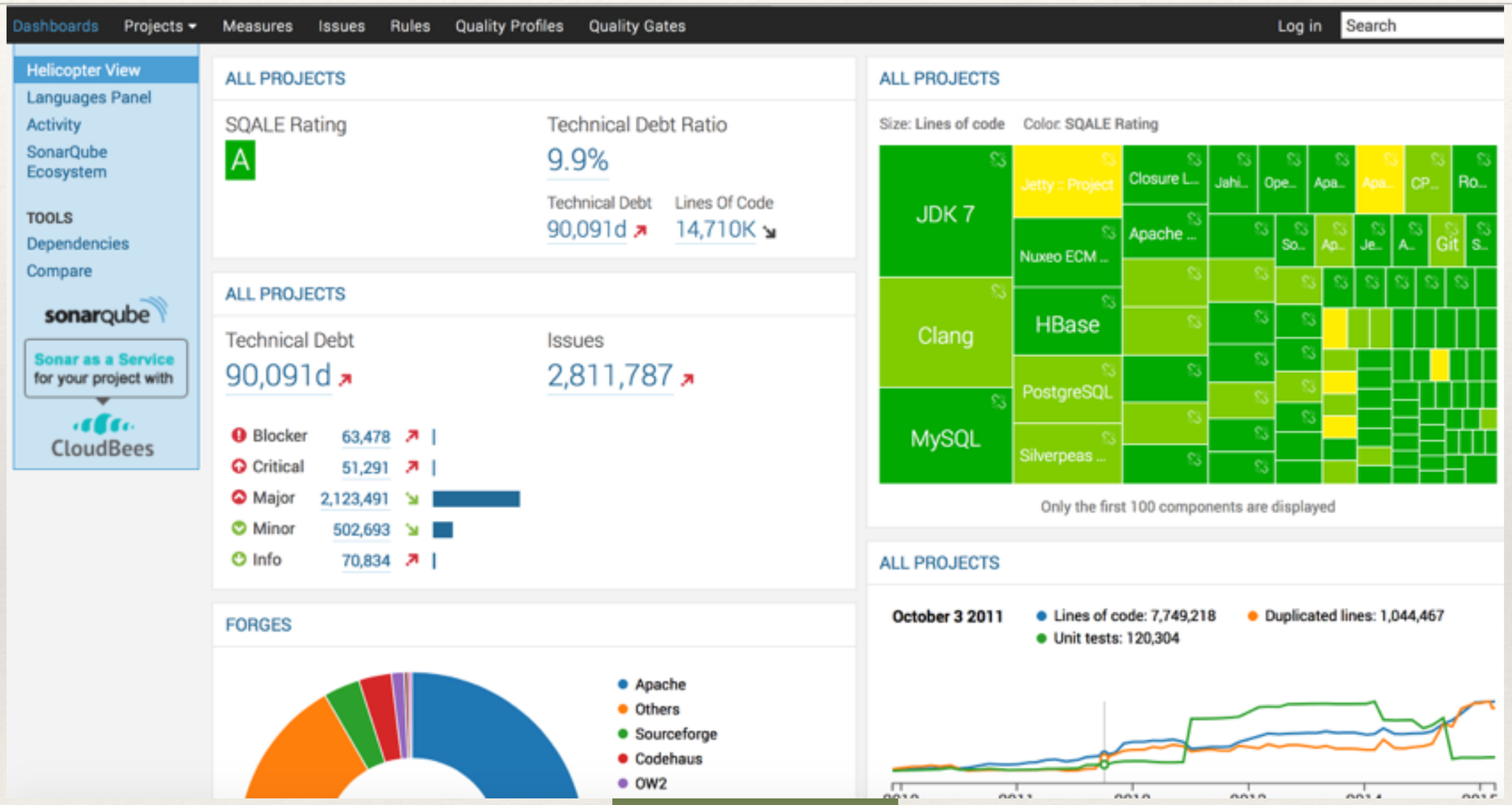
Metric tools

Understand

https://scitools.com



Technical debt quantification/visualization tools



Sonarqube

http://www.sonarqube.org

Refactoring tools

Contents

- Code Analysis
- Navigation and Search
- Coding Assistance
- Refactorings
- Project Level Features
- Code Generation
- Code Templates
- Code Cleanup
- Unit Testing
- Internationalization

ReSharper Features

The ultimate Agile tool is ReSharper. It is the one thing for .NET developers that removes fear of change. Refactoring is just so darn easy that change isn't scary.

Jaco Pretorius
ThoughtWorks

Code quality analysis

On-the-fly code quality analysis in C#, VB.NET, XAML, ASP.NET, ASP.NET MVC, JavaScript, TypeScript, CSS, HTML, and XML. ReSharper will tell you right away if your code contains errors or can be improved.

Eliminate errors and code smells

Instant fixes to eliminate errors and code smells. Not only does ReSharper warn you when there's a problem in your code but it provides quick-fixes to solve them automatically.

Instantly traverse your entire solution

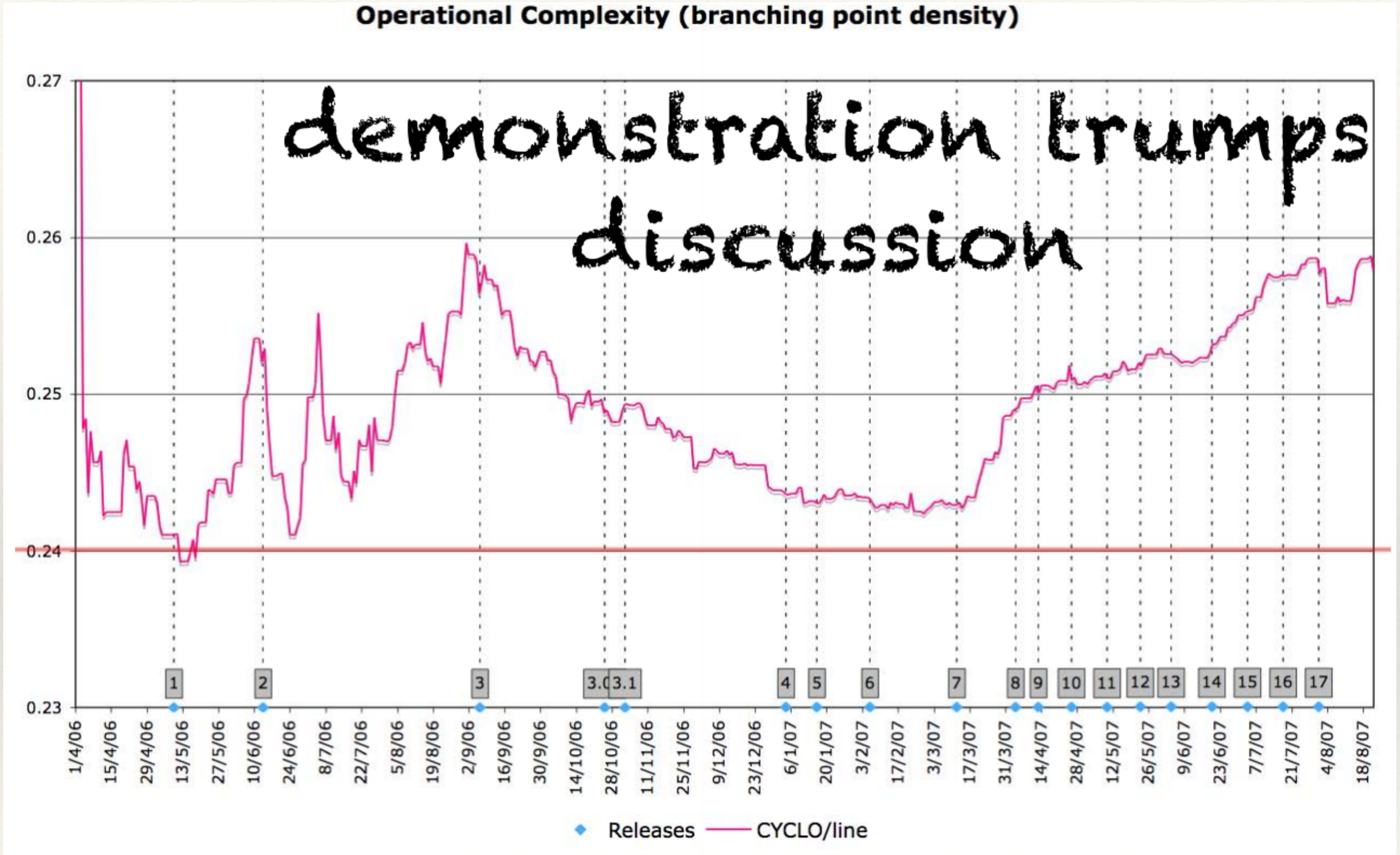
Navigation features to instantly traverse your entire solution. You can jump to any file, type, or member in your code base in no time, or navigate from a specific symbol to its usages, base and derived symbols, or implementations.

Safely change your code base

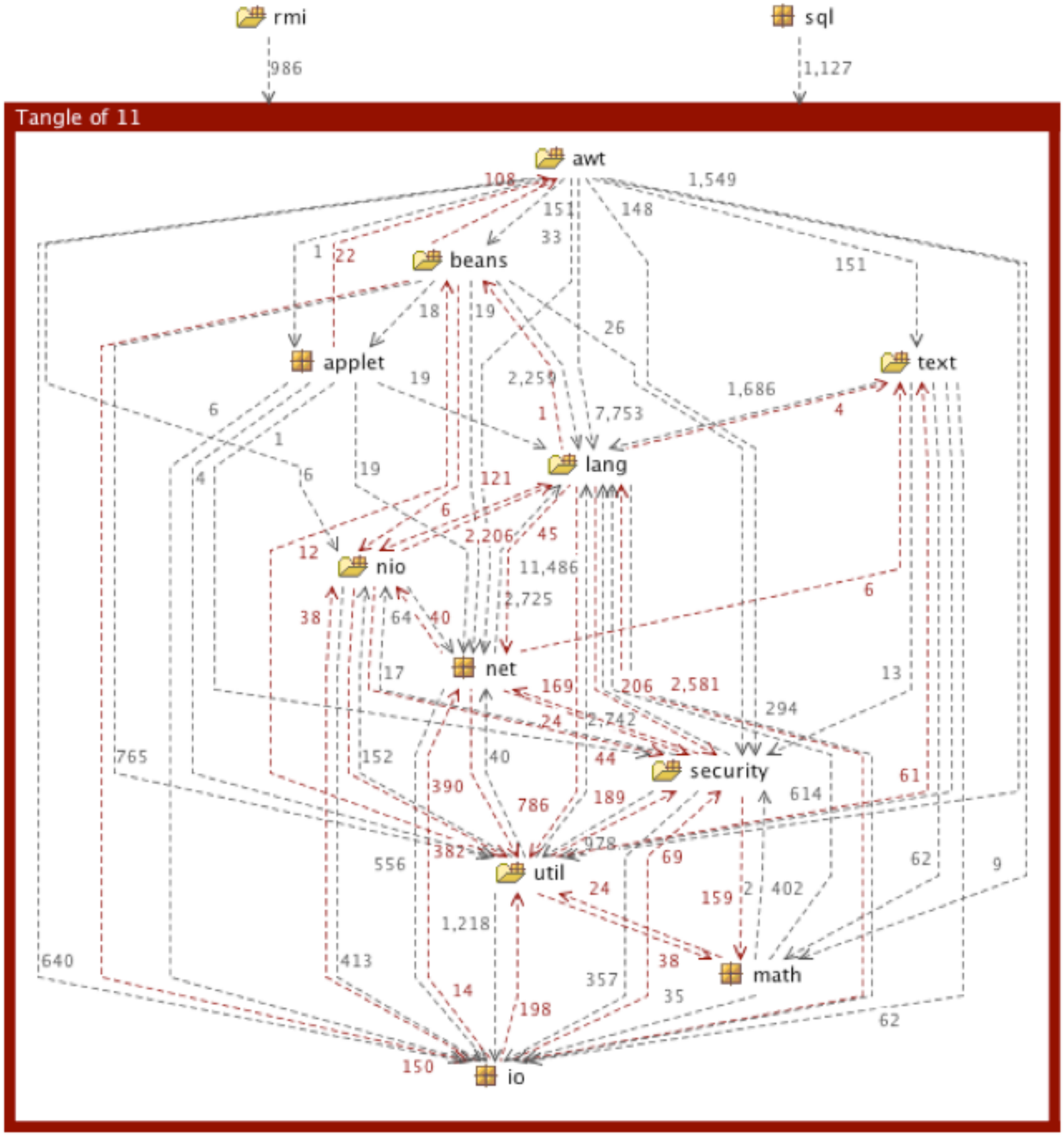
Enjoy solution-wide refactorings to safely change your code base. Whether you need to revitalize legacy code or put your project structure in order, you can lean on ReSharper.

ReSharper

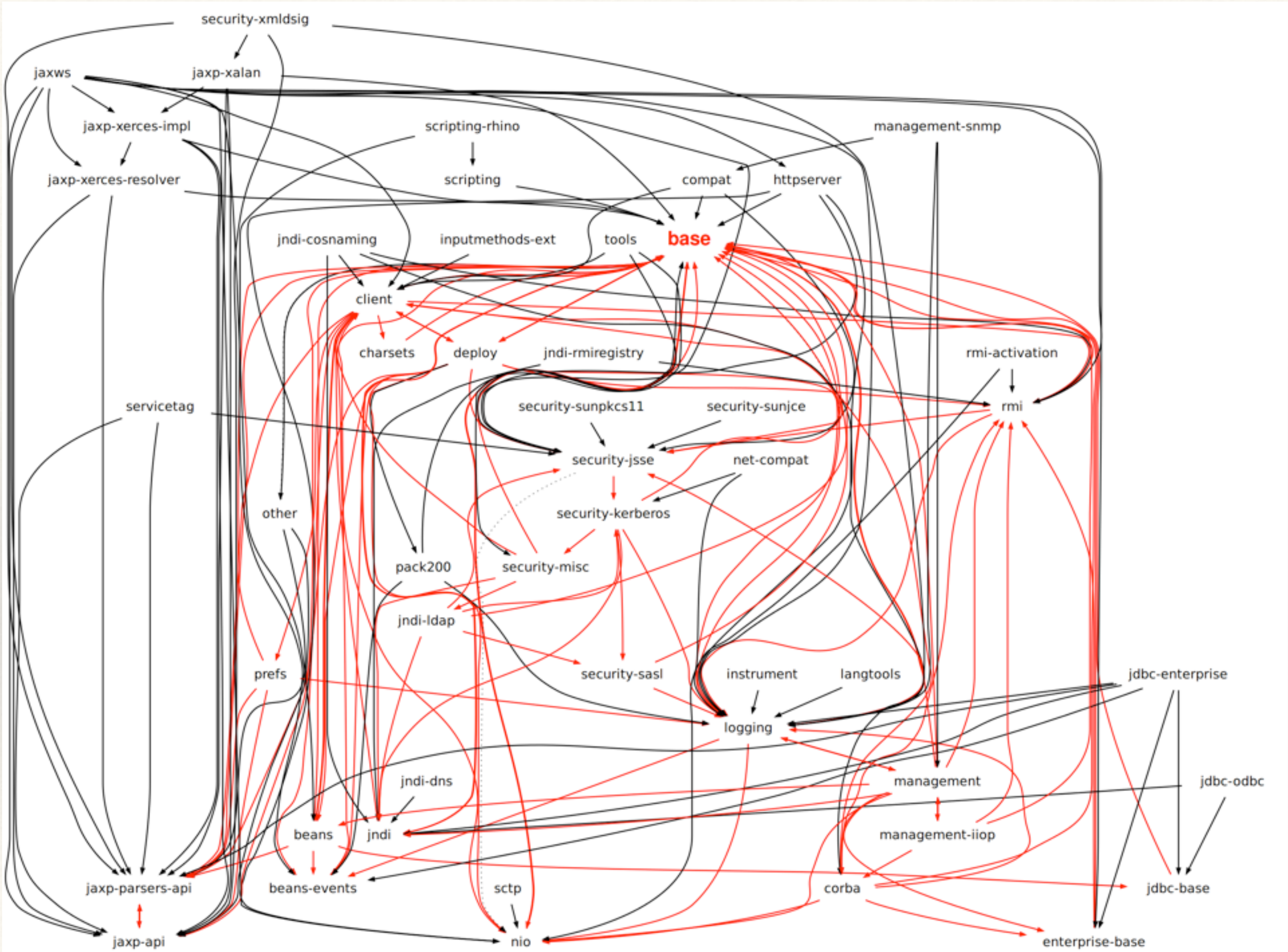
https://www.jetbrains.com/resharper/features/

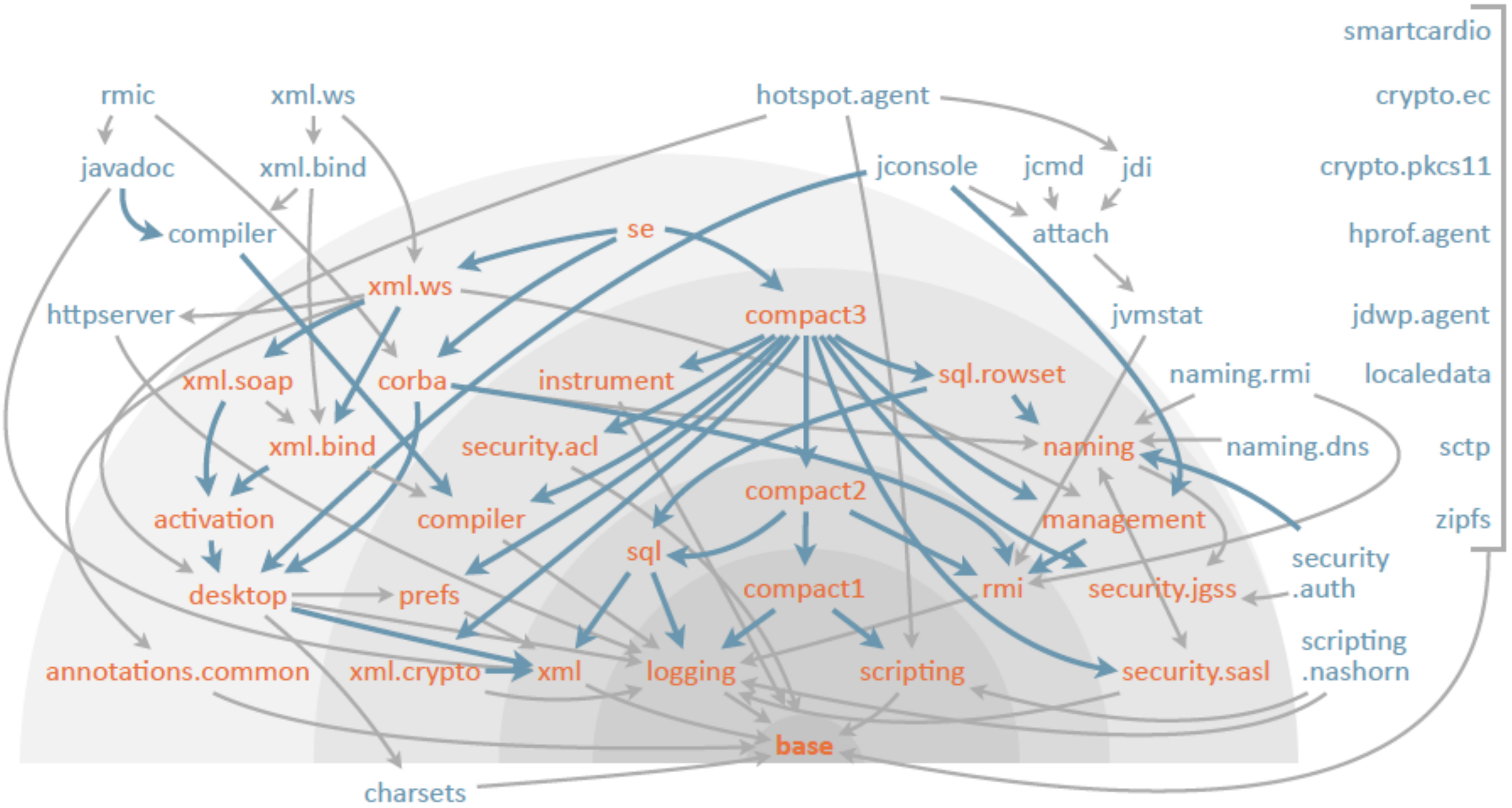


Source: Neal Ford, Emergent Design: https://dl.dropbox.com/u/6806810/Emergent_Design%28Neal_Ford%29.pdf



Tangles in JDK

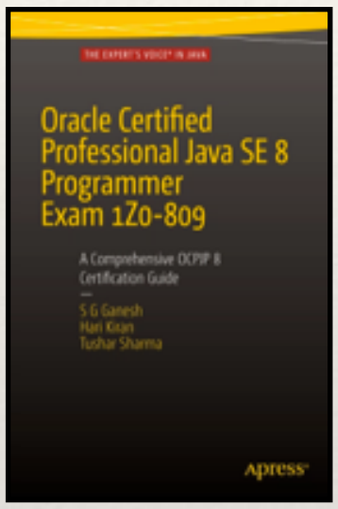

security-xmldsig
jaxws
jaxp-xalan
jaxp-xerces-impl
jaxp-xerces-resolver
scripting-rhino
scripting
management-snmp
compat
httpserver
jndi-cosnaming
inputmethods-ext
tools
base
client
charsets
deploy
jndi-rmiregistry
rmi-activation
rmi
servicetag
security-sunpkcs11
security-sunjce
security-jsse
net-compat
other
security-kerberos
pack200
security-misc
jndi-ldap
prefs
security-sasl
instrument
langtools
jdbc-enterprise
logging
jndi-dns
management
jdbc-odbc
beans
jndi
management-iiop
jaxp-parsers-api
beans-events
sctp
corba
jdbc-base
jaxp-api
nio
enterprise-base

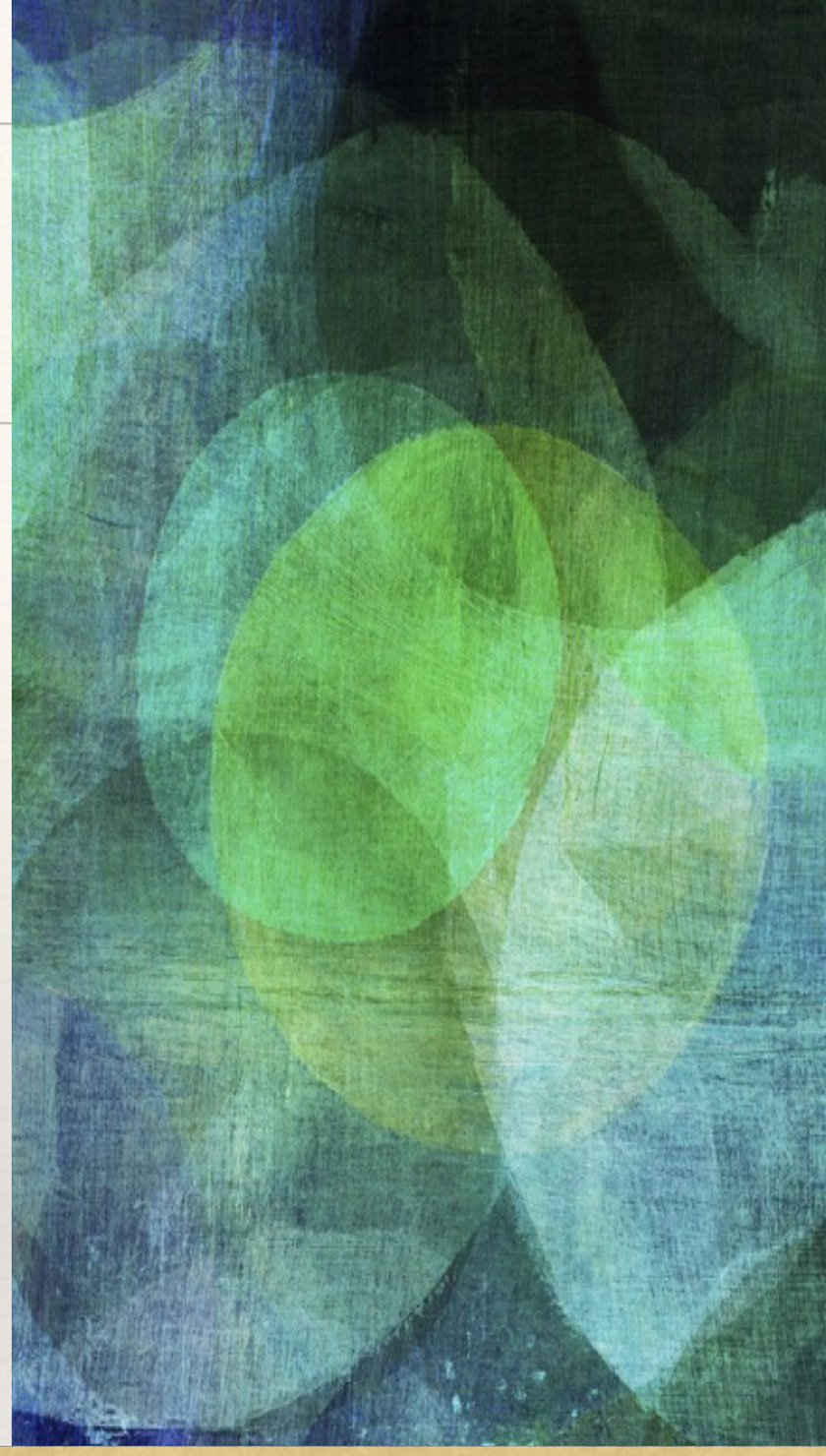

smartcardio
rmic
xml.ws
hotspot.agent
crypto.ec
javadoc
xml.bind
jconsole
jcmd
jdi
crypto.pkcs11
compiler
se
attach
hprof.agent
xml.ws
httpserver
compact3
jvmstat
jdwp.agent
xml.soap
corba
instrument
sql.rowset
naming.rmi
localedata
xml.bind
security.acl
naming
naming.dns
sctp
compact2
activation
compiler
management
zipfs
sql
security
.auth
desktop
prefs
compact1
rmi
security.jgss
scripting
.nashorn
annotations.common
xml.crypto
xml
logging
scripting
security.sasl
base
charsets

http://www.softrefactoring.com/

Refactoring for Software Architecture Smells

Ganesh Samarthyam, Tushar Sharma and Girish Suryanarayana





ganesh@codeops.tech

www.codeops.tech

+91 98801 64463

@GSamarthyam

slideshare.net/sgganesh

bit.ly/ganeshsg

CodeOps