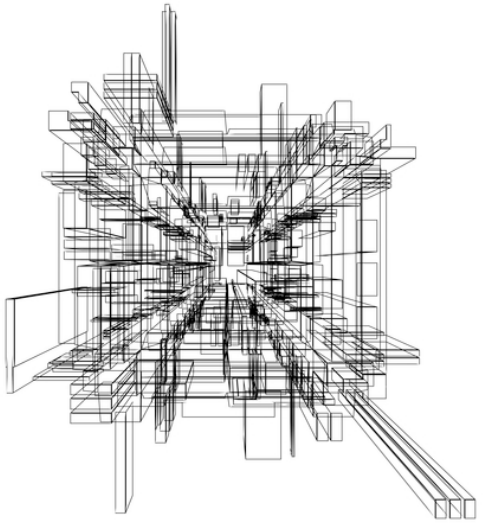
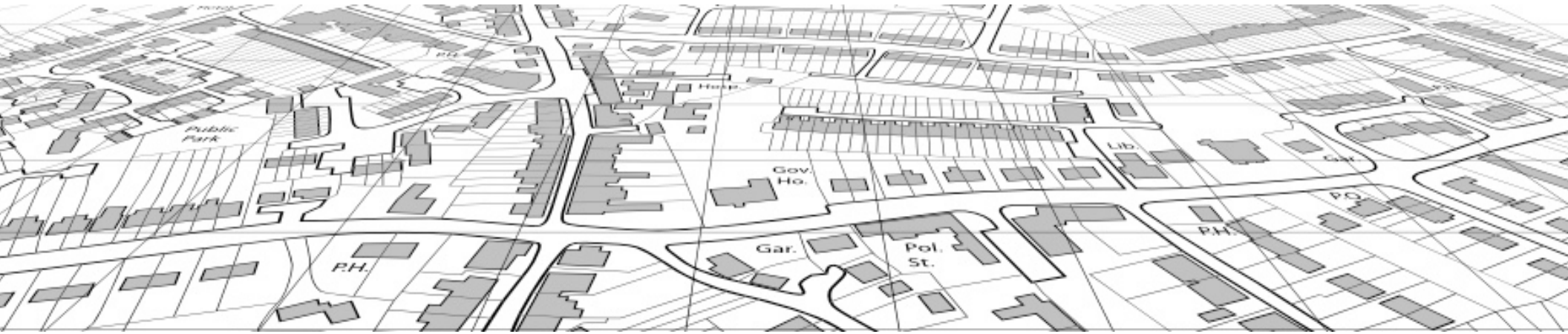


evolutionary architecture and emergent design



Rising or emerging out of a fluid or anything that covers or conceals; issuing; coming to light. [1913 Webster]

Emergente

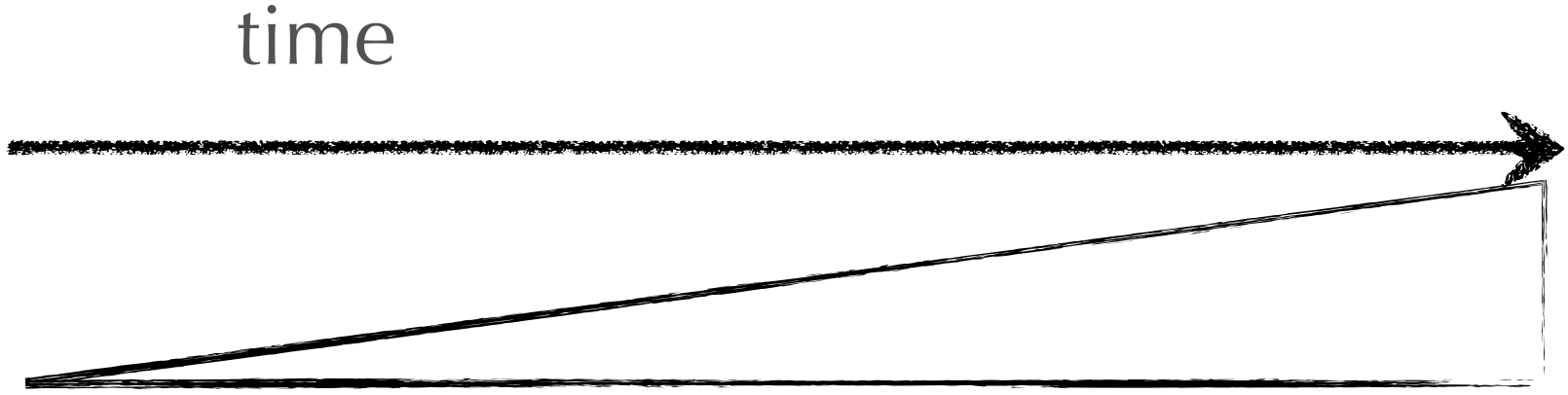
Suddenly appearing; arising unexpectedly; calling for prompt action; urgent. [1913 Webster]


finding
abstractions &
patterns

finding abstractions & patterns

last responsible moment

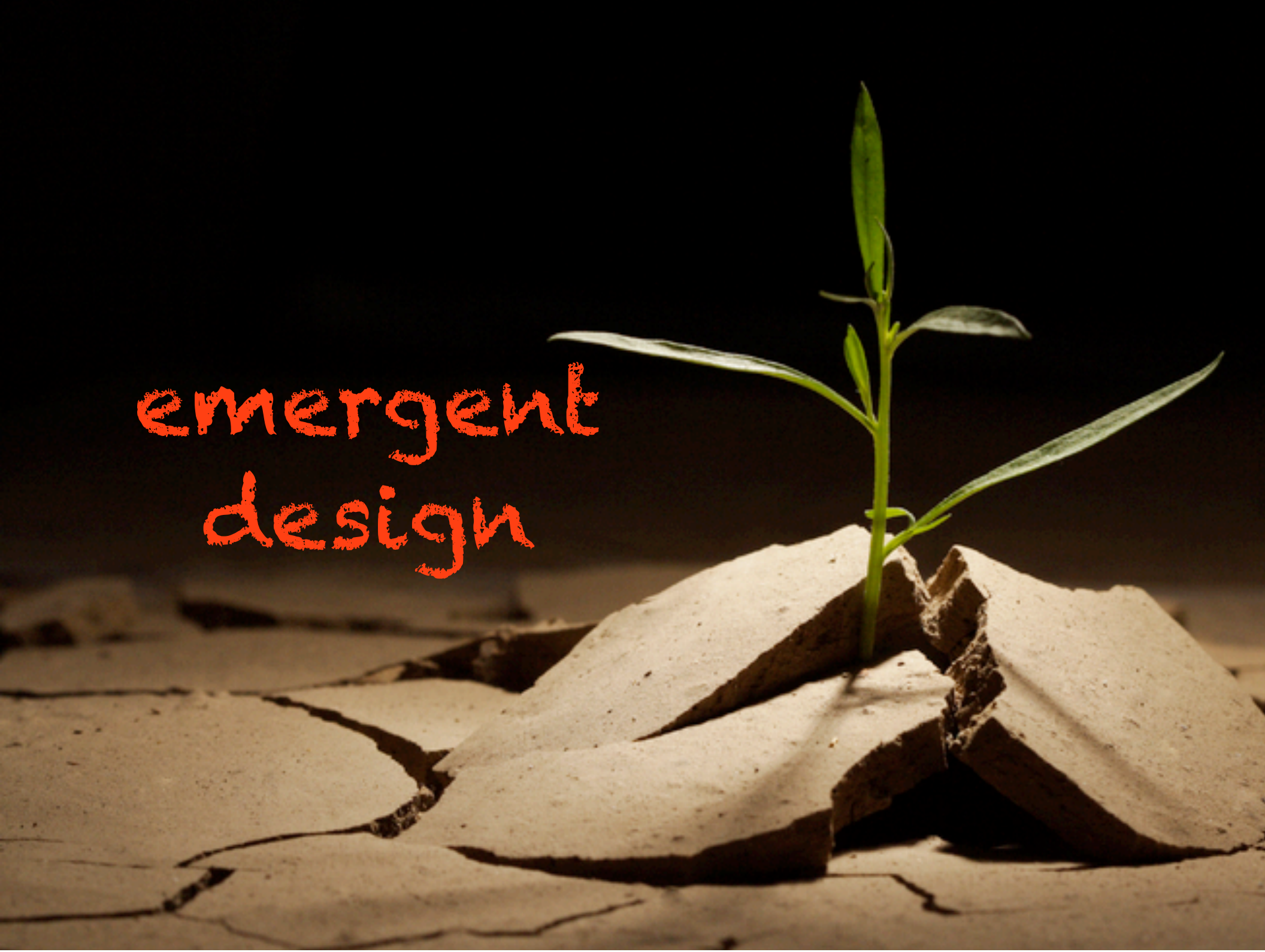
Last responsible moment



longer delay = more real data for decision

finding abstractions & patterns

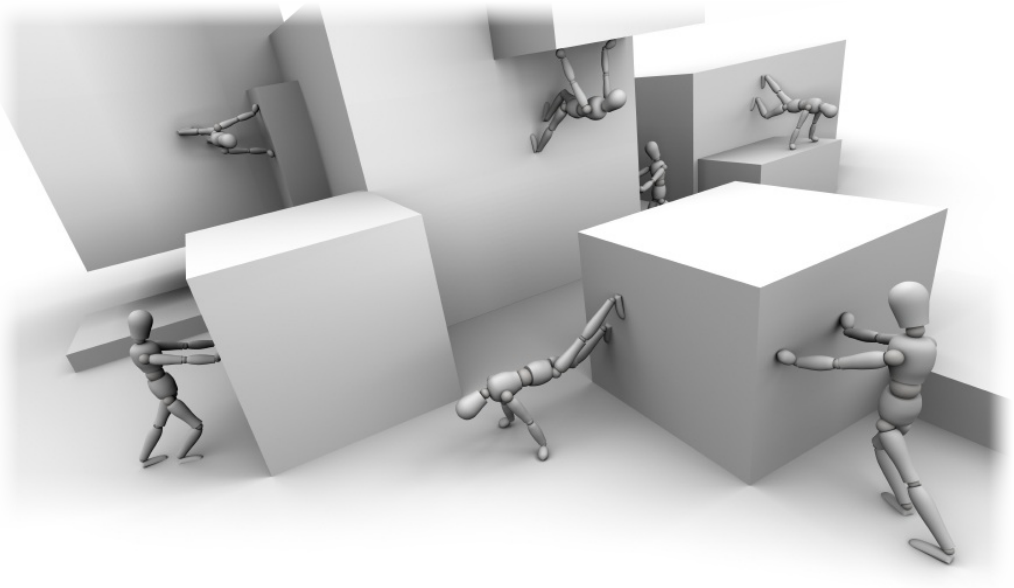
last responsible moment


emergent
design

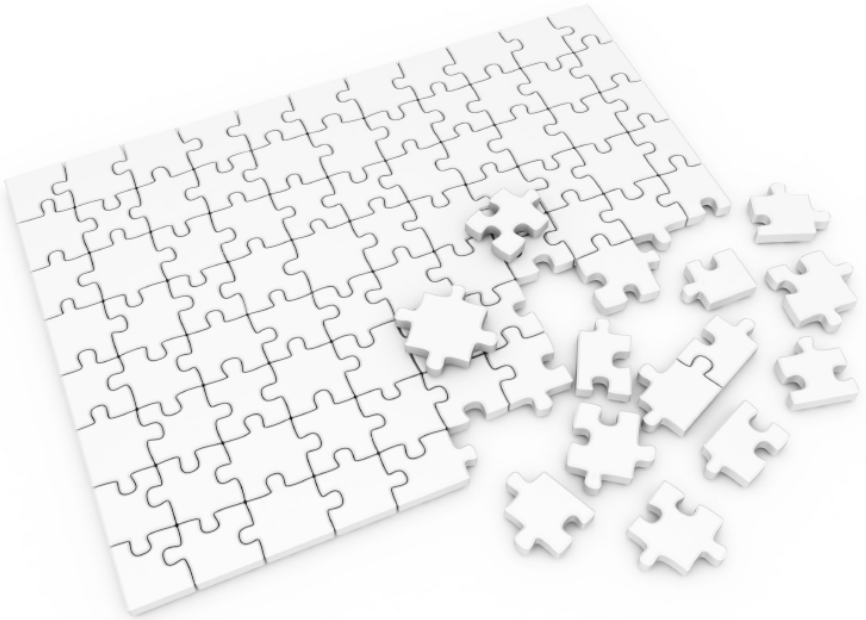
complexity

accidental complexity

all the externally imposed ways software becomes complex



VS

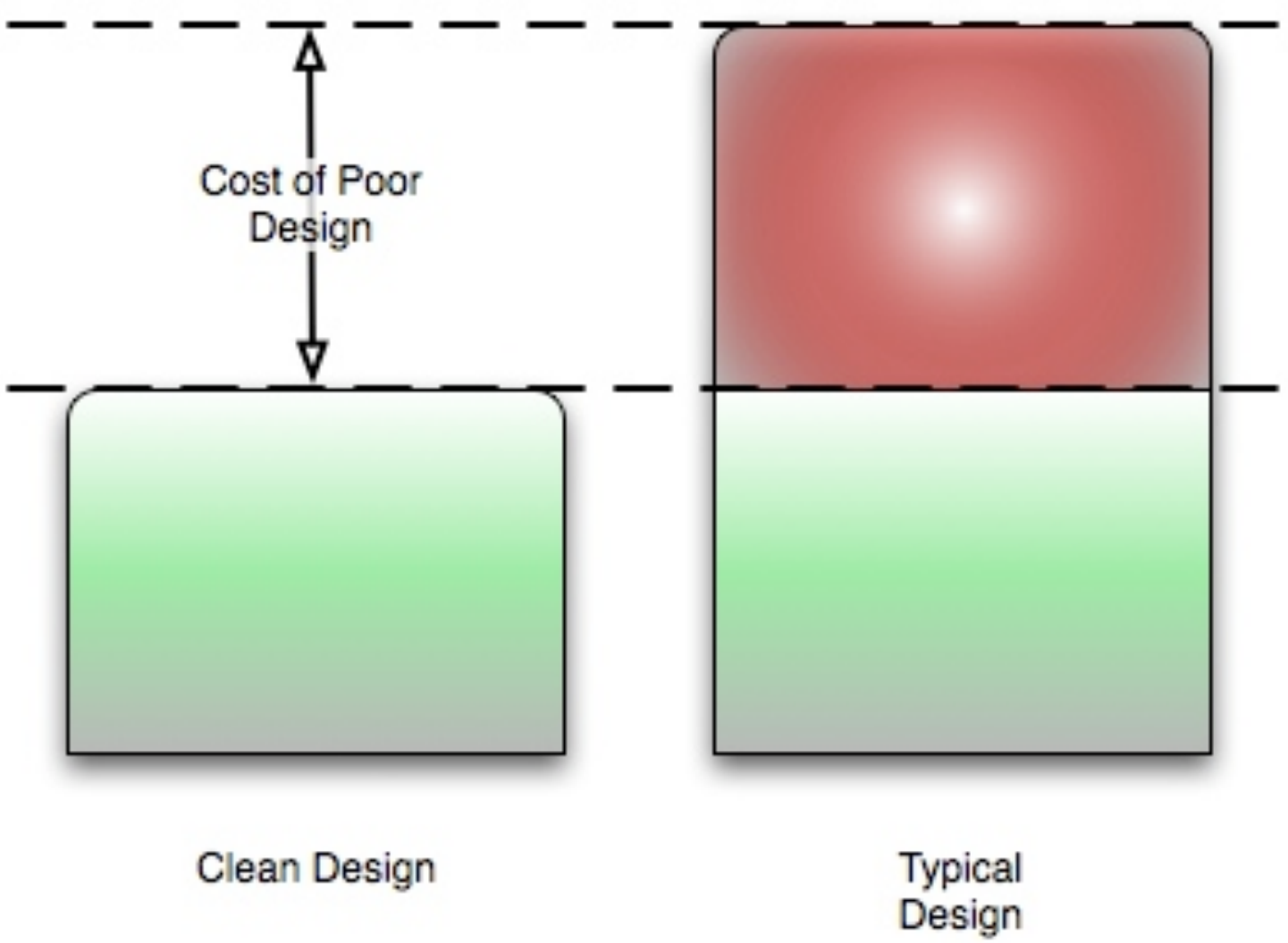


essential complexity

inherent complexity


technical
debt

technical debt

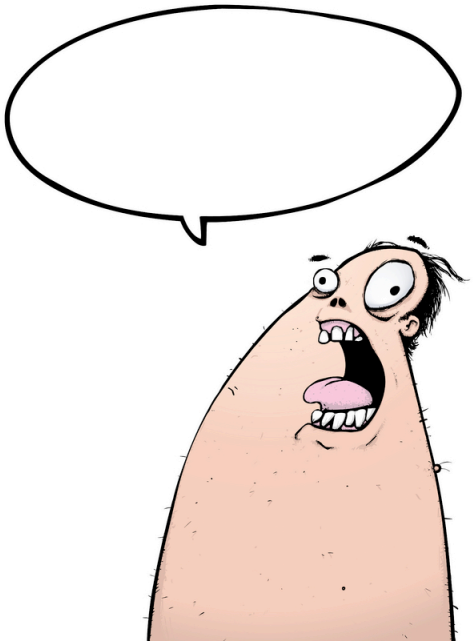


reckless	**prudent**
"We don't have time for design."	*"We must ship now & deal with the consequences."*
deliberate	
inadvertent	
"What's layering?"	*"Now we know how we should have done it."*

http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

negotiating repayment

demonstration

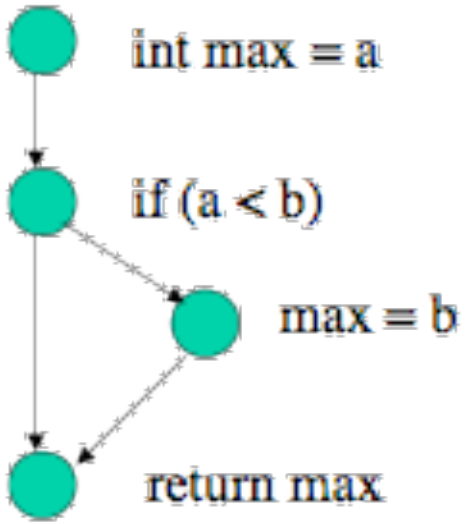


cyclomatic complexity

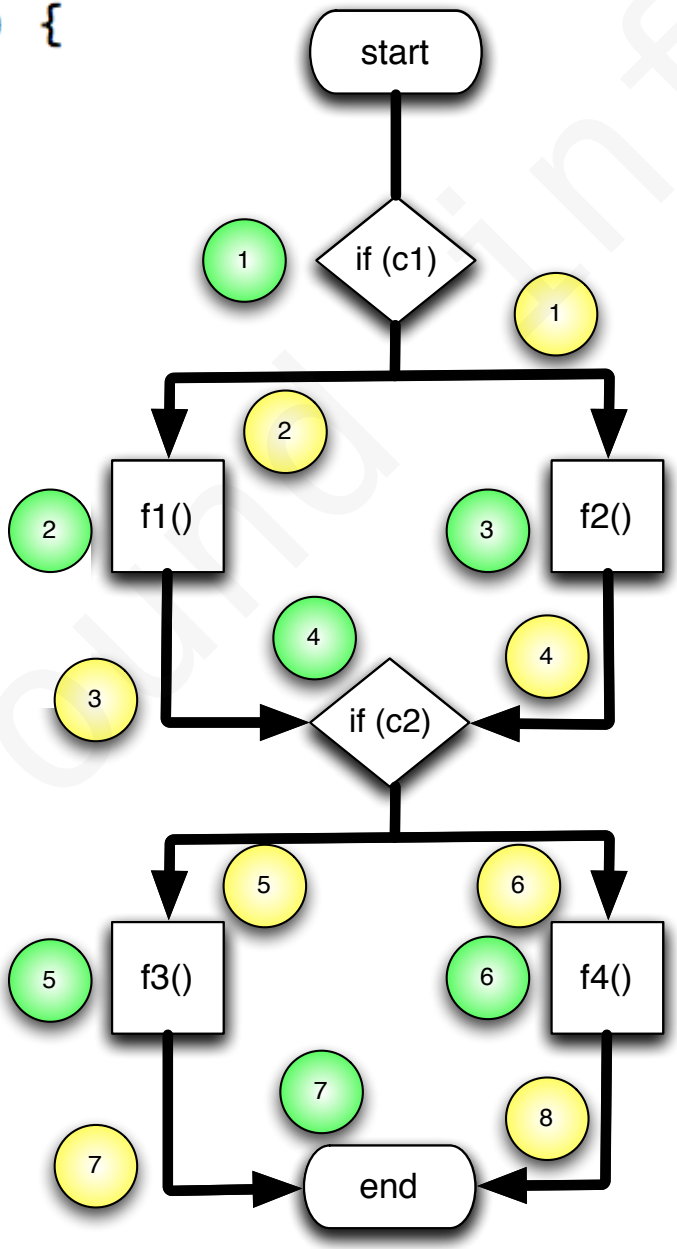
measures complexity of a method/function

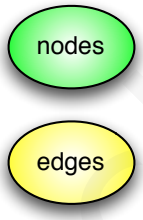
```
V(G)= e - n + 2
V(G) = cyclomatic complexity of G
e= # edges
n= # of nodes
```

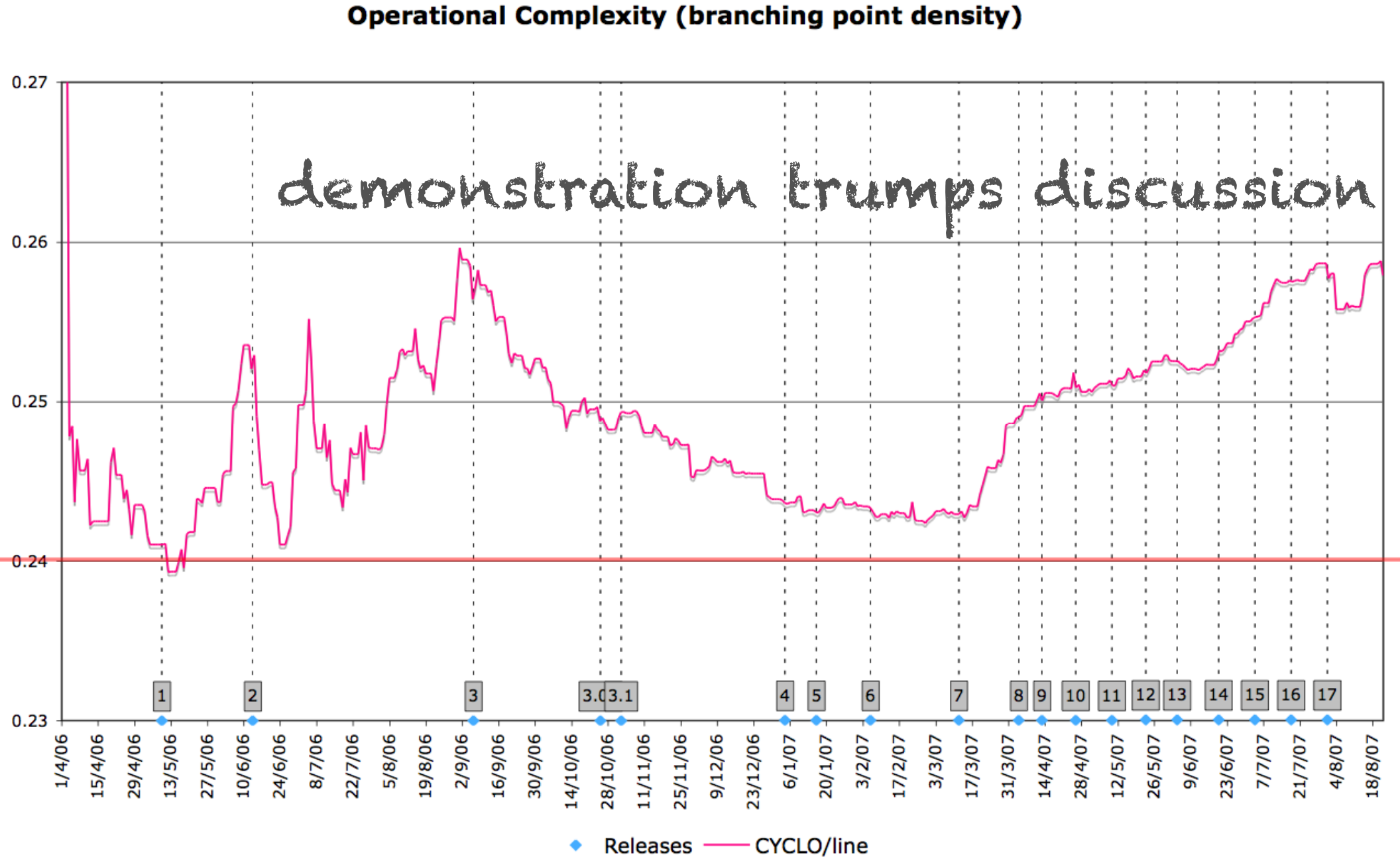
```
int max (int a, int b) {
    int max = a;
    if (a < b) {
        max = b;
    }
    return max;
}
```



```
public void doIt() {
    if (c1) {
        f1();
    } else {
        f2();
    }
    if (c2) {
        f3();
    } else {
        f4();
    }
}
```






Operational Complexity (branching point density)
demonstration trumps discussion
0.27
0.26
0.25
0.24
0.23
1
2
3
3.0
3.1
4
5
6
7
8
9
10
11
12
13
14
15
16
17
1/4/06
15/4/06
29/4/06
13/5/06
27/5/06
10/6/06
24/6/06
8/7/06
22/7/06
5/8/06
19/8/06
2/9/06
16/9/06
30/9/06
14/10/06
28/10/06
11/11/06
25/11/06
9/12/06
23/12/06
6/1/07
20/1/07
3/2/07
17/2/07
3/3/07
17/3/07
31/3/07
14/4/07
28/4/07
12/5/07
26/5/07
9/6/07
23/6/07
7/7/07
21/7/07
4/8/07
18/8/07
Releases
CYCLO/line

Sonar – Struts 2

Home | Search | Struts 2 » | Configuration | Log In

- Dashboard
- Components
- Violations drilldown
- Time machine
- Clouds
- Design
- Hotspots
- Motion chart
- Radiator
- Timeline

sonar

Version 2.2.0-SNAPSHOT - Tue, 09 Feb 2010 17:17 - profile Nemo rules with findbugs

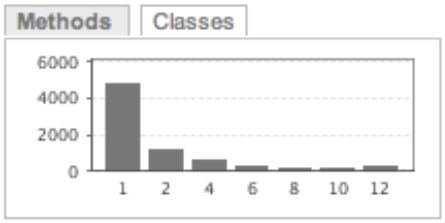
Lines of code
79,396
147,345 lines

Classes
1,145
131 packages
7,323 methods
+936 accessors

Comments
20.1%
19,988 lines
30.7% docu. API
4,800 undocu. API
143 commented LOCs

Duplications
3.0%
4,432 lines
150 blocks
66 files

Complexity
2.7 / method
17.4 / class
19,908 cmpx
38,241 statements



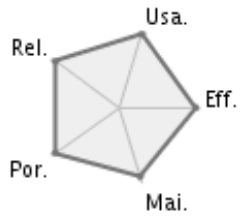
Code coverage
26.2%
28.2% line coverage
21.7% branch coverage
1,126 tests
1:46 min

Test success
99.8%
2 failures
0 errors

Events All

2010-02-09	Version	2.2.0-SNAPSHOT
2009-09-28	Version	2.1.9-SNAPSHOT

Rules compliance
86.9%

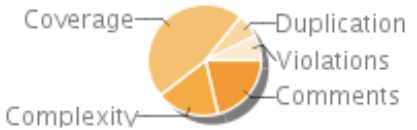


Violations
3,833

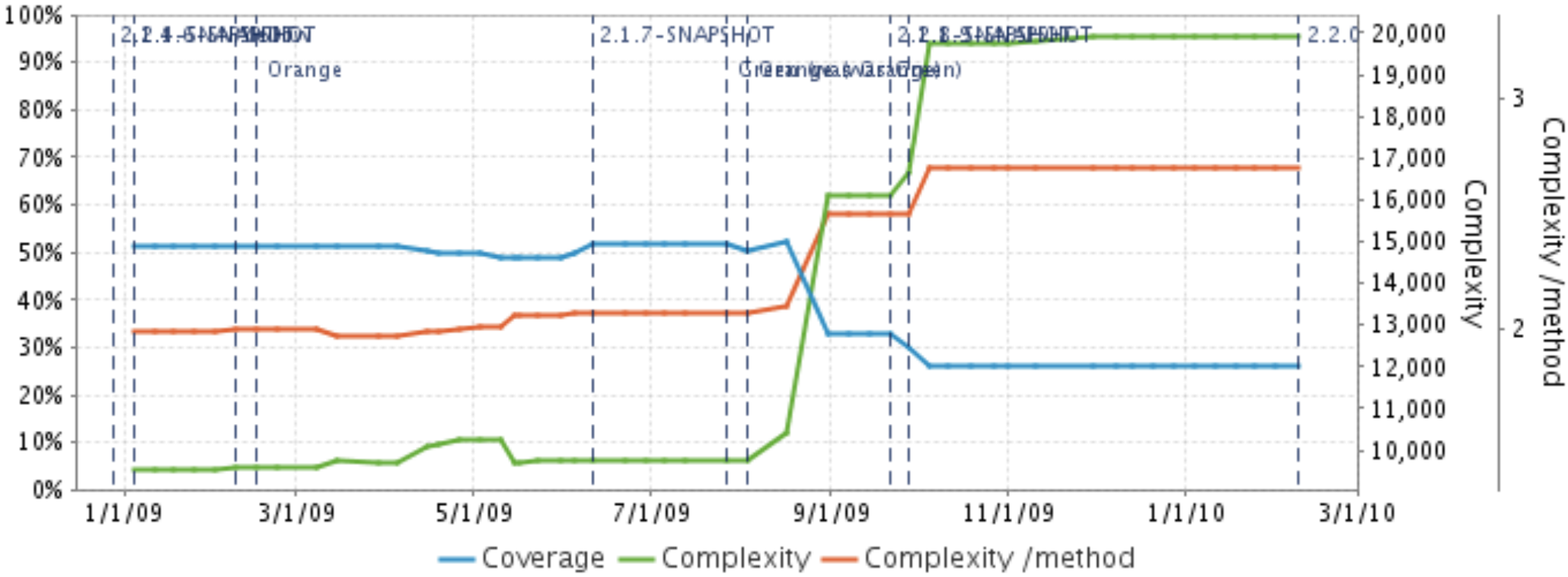
Blocker	0
Critical	0
Major	3,460
Minor	0
Info	373

Alerts : Unit test success (%) < 100.

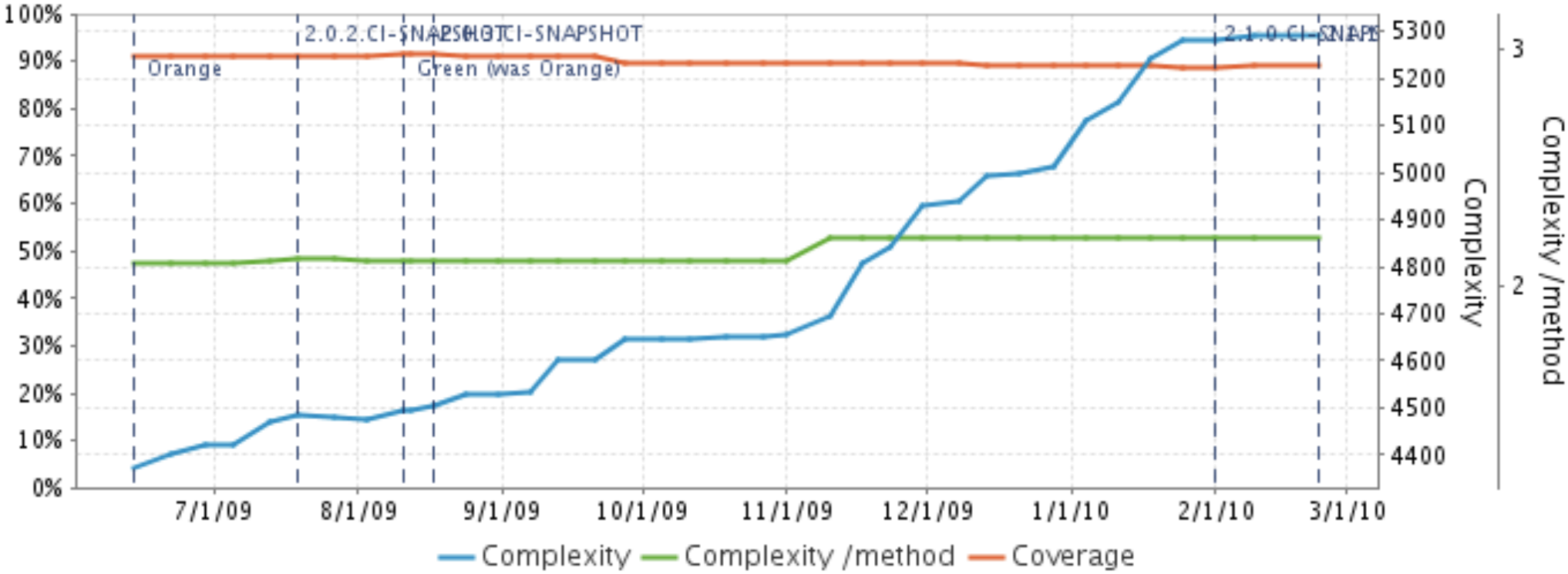
Technical Debt
15.2%
$ 286,100
572 man days

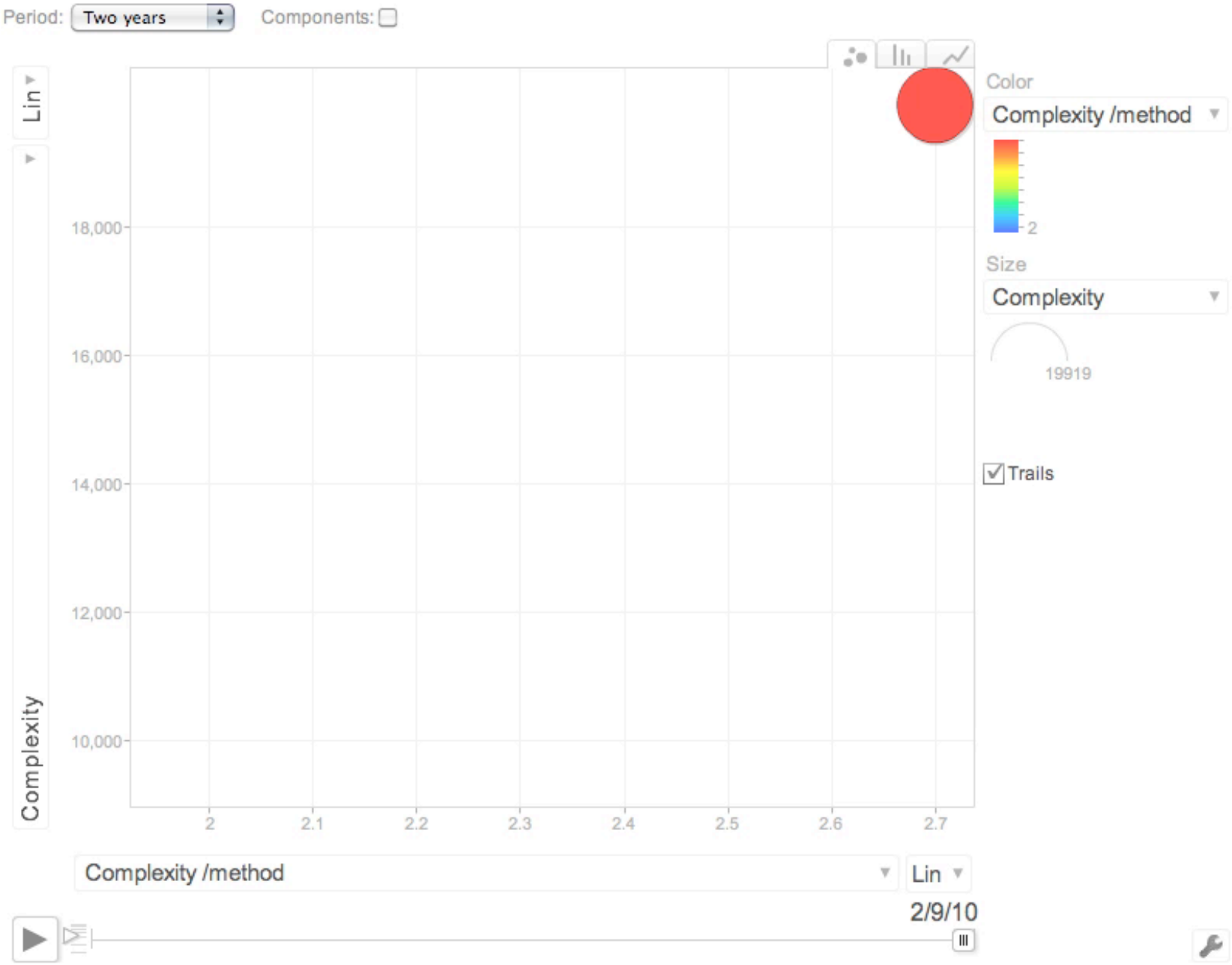


time machine (struts)

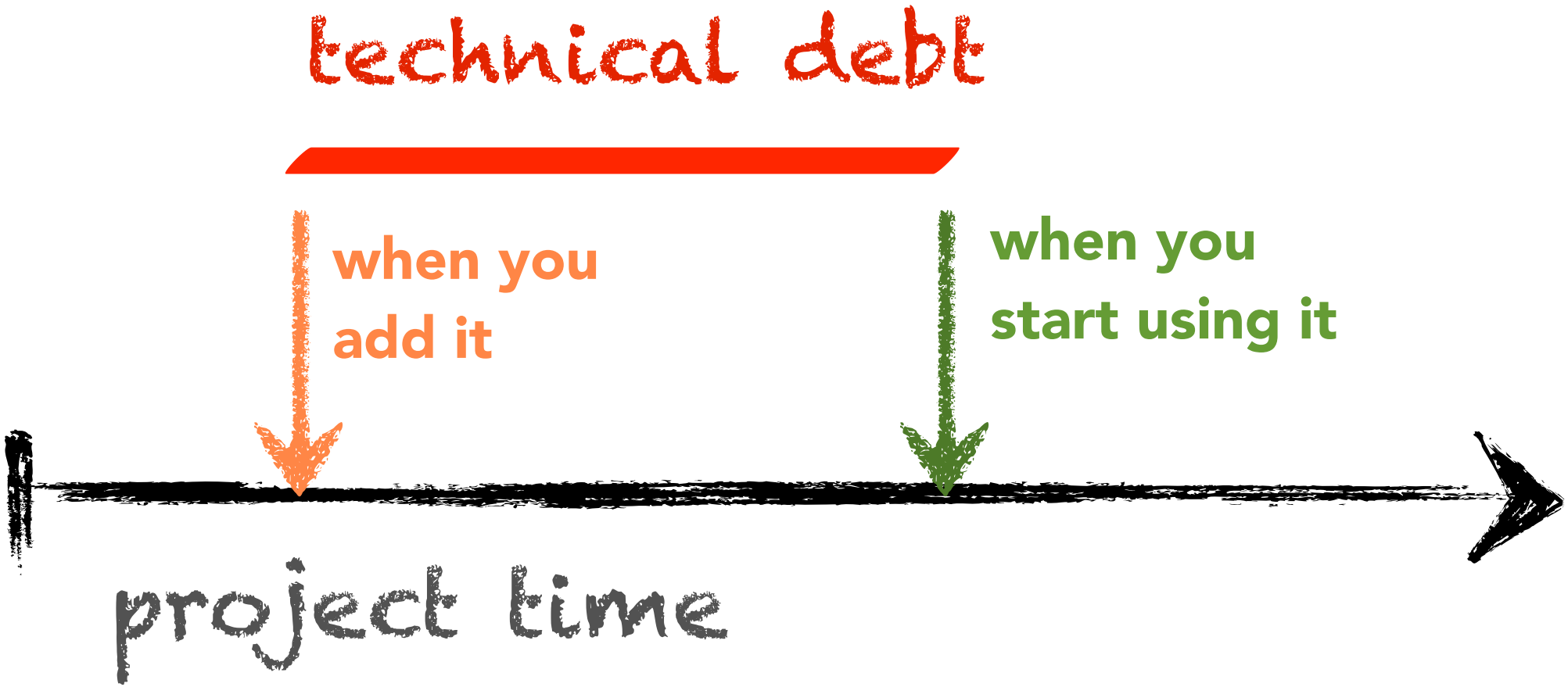


time machine (spring batch)




Period: Two years
Components:
Lin
18,000
16,000
14,000
12,000
10,000
Complexity
2
2.1
2.2
2.3
2.4
2.5
2.6
2.7
Color
Complexity /method
2
Size
Complexity
19919
Trails
Complexity /method
Lin
2/9/10

premature optimization



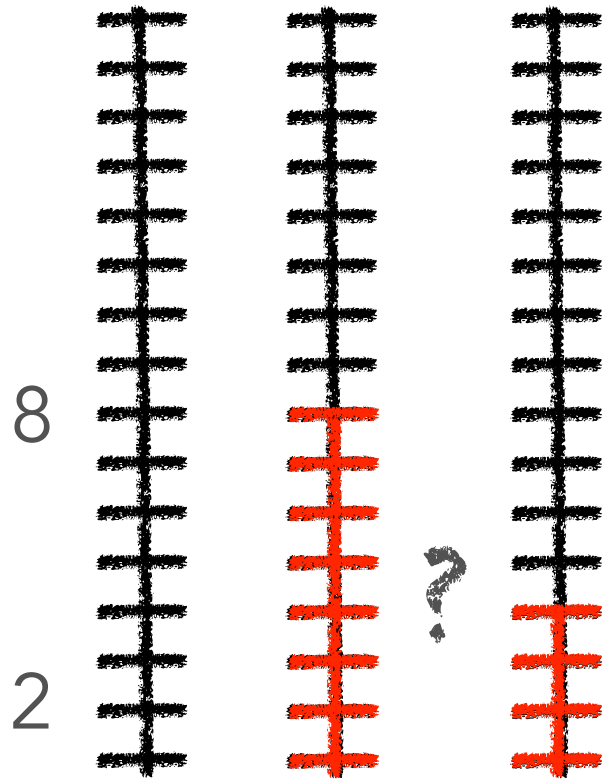
Emergent Design Accelerators

perfect number case study

∑ of the factors == number
(not including the number)

test-after, 1st pass

```
public class PerfectNumberFinder1 {
    public static boolean isPerfect(int number) {
        // get factors
        List<Integer> factors = new ArrayList<Integer>();
        factors.add(1);
        factors.add(number);
        for (int i = 2; i < number; i++)
            if (number % i == 0)
                factors.add(i);

        // sum factors
        int sum = 0;
        for (int n : factors)
            sum += n;

        // decide if it's perfect
        return sum - number == number;
    }
}
```

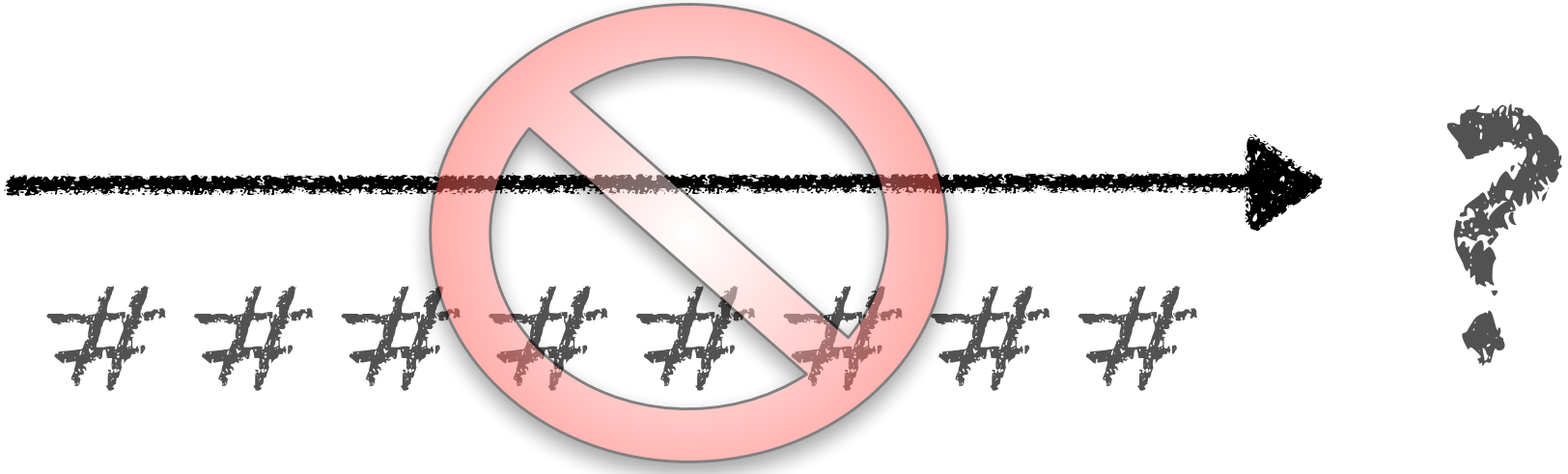


```
public class PerfectNumberFinder2 {
    public static boolean isPerfect(int number) {
        // get factors
        List<Integer> factors = new ArrayList<Integer>();
        factors.add(1);
        factors.add(number);
        for (int i = 2; i <= sqrt(number); i++)
            if (number % i == 0) {
                factors.add(i);
                factors.add(number / i);
            }

        // sum factors
        int sum = 0;
        for (int n : factors)
            sum += n;

        // decide if it's perfect
        return sum - number == number;
    }
}
```

whole-number square roots

```
public class PerfectNumberFinder2 {
    public static boolean isPerfect(int number) {
        // get factors
        List<Integer> factors = new ArrayList<Integer>();
        factors.add(1);
        factors.add(number);
        for (int i = 2; i <= sqrt(number); i++)
            if (number % i == 0) {
                factors.add(i);
                // guard against whole-number square roots
                if (number / i !=  i)
                    factors.add(number / i);
            }

        // sum factors
        int sum = 0;
        for (int n : factors)
            sum += n;

        // decide if it's perfect
        return sum - number == number;
    }
}
```

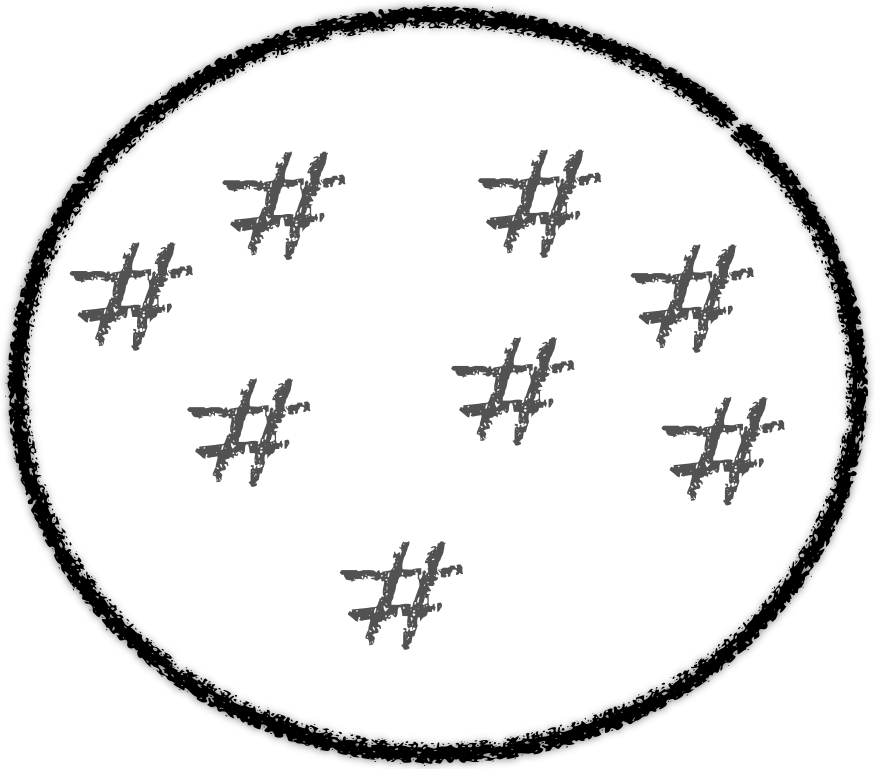
```
public class Classifier6 {
    private Set<Integer> _factors;
    private int _number;

    public Classifier6(int number) {
        if (number < 1)
            throw new InvalidNumberException(
            "Can't classify negative numbers");
        _number = number;
        _factors = new HashSet<Integer>();
        _factors.add(1);
        _factors.add(_number);
    }

    private boolean isFactor(int factor) {
        return _number % factor == 0;
    }

    public Set<Integer> getFactors() {
        return _factors;
    }

    private void calculateFactors() {
        for (int i = 2; i < sqrt(_number) + 1; i++)
            if (isFactor(i))
                addFactor(i);
    }

    private void addFactor(int factor) {
        _factors.add(factor);
        _factors.add(_number / factor);
    }

    private int sumOfFactors() {
        calculateFactors();
        int sum = 0;
        for (int i : _factors)
            sum += i;
        return sum;
    }

    public boolean isPerfect() {
        return sumOfFactors() - _number == _number;
    }
```



Done: 1 of 1 Failed: 1(0.035 s)

Output | Statistics

```
java.lang.AssertionError:
Expected: is <[1, 2, 3, 6]>
     got: <[1, 6, 2, 3]>

    at org.junit.Assert.assertThat(Assert.java:502)
    at org.junit.Assert.assertThat(Assert.java:492)
    at com.nealford.conf.tdd.perfectnumbers.Classifier3Test
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMe
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Dele
    at org.junit.internal.runners.TestMethod.invoke(TestMet
    at org.junit.internal.runners.MethodRoadie.runTestMetho
    at org.junit.internal.runners.MethodRoadie$2.run(Method
```



test-after

```
for (int i = 2; i <= sqrt(number); i++)
    if (number % i == 0) {
        factors.add(i);
        // account for whole-number square roots
        if (number / i !=  i)
            factors.add(number / i);
    }
```

TDD

```
private void calculateFactors() {
    for (int i = 2; i < sqrt(_number) + 1; i++)
        if (isFactor(i))
            addFactor(i);
}

private void addFactor(int factor) {
    _factors.add(factor);
    _factors.add(_number / factor);
}
```

```
public void addOrder(final ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection c = null; PreparedStatement ps = null;
    Statement s = null; ResultSet rs = null;
    boolean transactionState = false;
    try {
        c = dbPool.getConnection();
        s = c.createStatement();
        transactionState = c.getAutoCommit();
        int userKey = getUserKey(userName, c, ps, rs);
        c.setAutoCommit(false);
        addSingleOrder(order, c, ps, userKey);
        int orderKey = getOrderKey(s, rs);
        addLineItems(cart, c, orderKey);
        c.commit();
        order.setOrderKey(orderKey);
    } catch (SQLException sqlx) {
        s = c.createStatement();
        c.rollback();
        throw sqlx;
    } finally {
        try {
            c.setAutoCommit(transactionState);
            dbPool.release(c);
            if (s != null)  s.close();
            if (ps != null) ps.close();
            if (rs != null) rs.close();
        } catch (SQLException ignored) {
        }
    }
}
```

refactoring towards design

```
public void addOrder(final ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection connection = null; PreparedStatement ps = null;
    Statement statement = null; ResultSet rs = null;
    boolean transactionState = false;
    try {
        connection = dbPool.getConnection();
        statement = connection.createStatement();
        transactionState = setupTransactionStateFor(connection, transactionState);
        addSingleOrder(order, connection, ps, userKeyFor(userName, connection));
        order.setOrderKey(generateOrderKey(statement, rs));
        addLineItems(cart, connection, order.getOrderKey());
        completeTransaction(connection);
    } catch (SQLException sqlx) {
        rollbackTransactionFor(connection);
        throw sqlx;
    } finally {
        cleanUpDatabaseResources(connection, transactionState, statement, ps, rs);
    }
}
```

idiomatic “unit of work” pattern

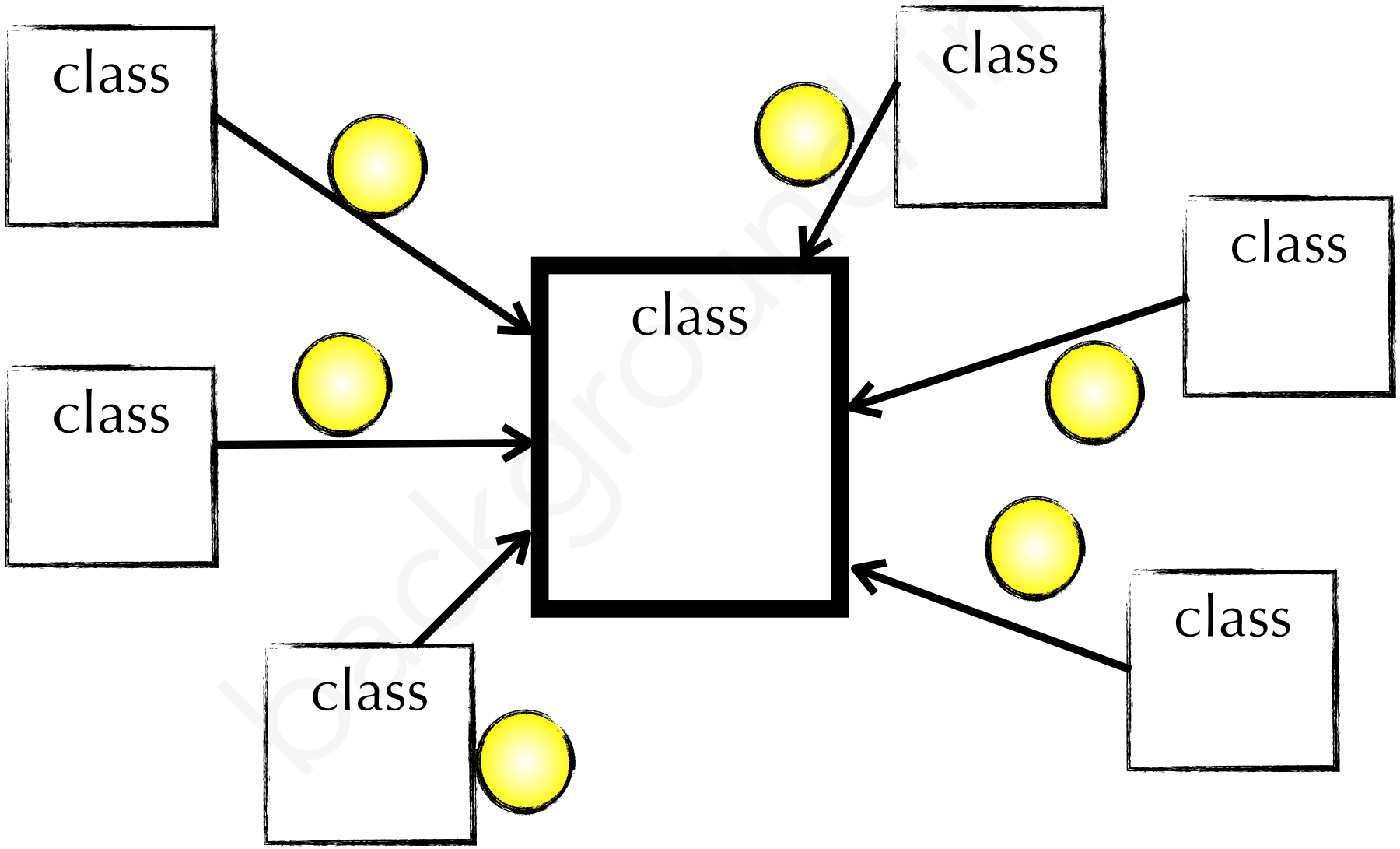
```
public void addOrderFrom(ShoppingCart cart, String userName,
                         Order order) throws SQLException {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
        completeTransaction();
    } catch (SQLException sqlx) {
        rollbackTransaction();
        throw sqlx;
    } finally {
        cleanUp();
    }
}
```

see the *composed method* pattern

Smalltalk Best Practice Patterns Kent Beck

Refactoring to Harvest Idiomatic Patterns

afferent coupling



classname	WMC	Ca
org.apache.struts2.components.Component	28	177
org.apache.struts2.views.freemarker.tags.TagModel	7	47
org.apache.struts2.views.velocity.components.AbstractDirective	8	43
org.apache.struts2.StrutsException	7	23
org.apache.struts2.components.UIBean	53	22
org.apache.struts2.dispatcher.mapper.ActionMapping	13	20
org.apache.struts2.views.jsp.ComponentTagSupport	6	19
org.apache.struts2.dispatcher.Dispatcher	37	19
org.apache.struts2.views.jsp.ui.AbstractUITag	34	18
org.apache.struts2.views.xslt.AdapterFactory	9	16
org.apache.struts2.views.xslt.AdapterNode	10	15
org.apache.struts2.ServletActionContext	11	15
org.apache.struts2.components.table.WebTable	33	12
org.apache.struts2.dispatcher.mapper.ActionMapper	2	11
org.apache.struts2.components.template.TemplateEngine	2	10
org.apache.struts2.components.template.Template	7	10
org.apache.struts2.dispatcher.StrutsResultSupport	13	10
org.apache.struts2.components.Form	24	10
org.apache.struts2.components.ListUIBean	8	9
org.apache.struts2.util.MakeIterator	3	8
org.apache.struts2.StrutsStatics	0	7

UIBean evaluateParams()

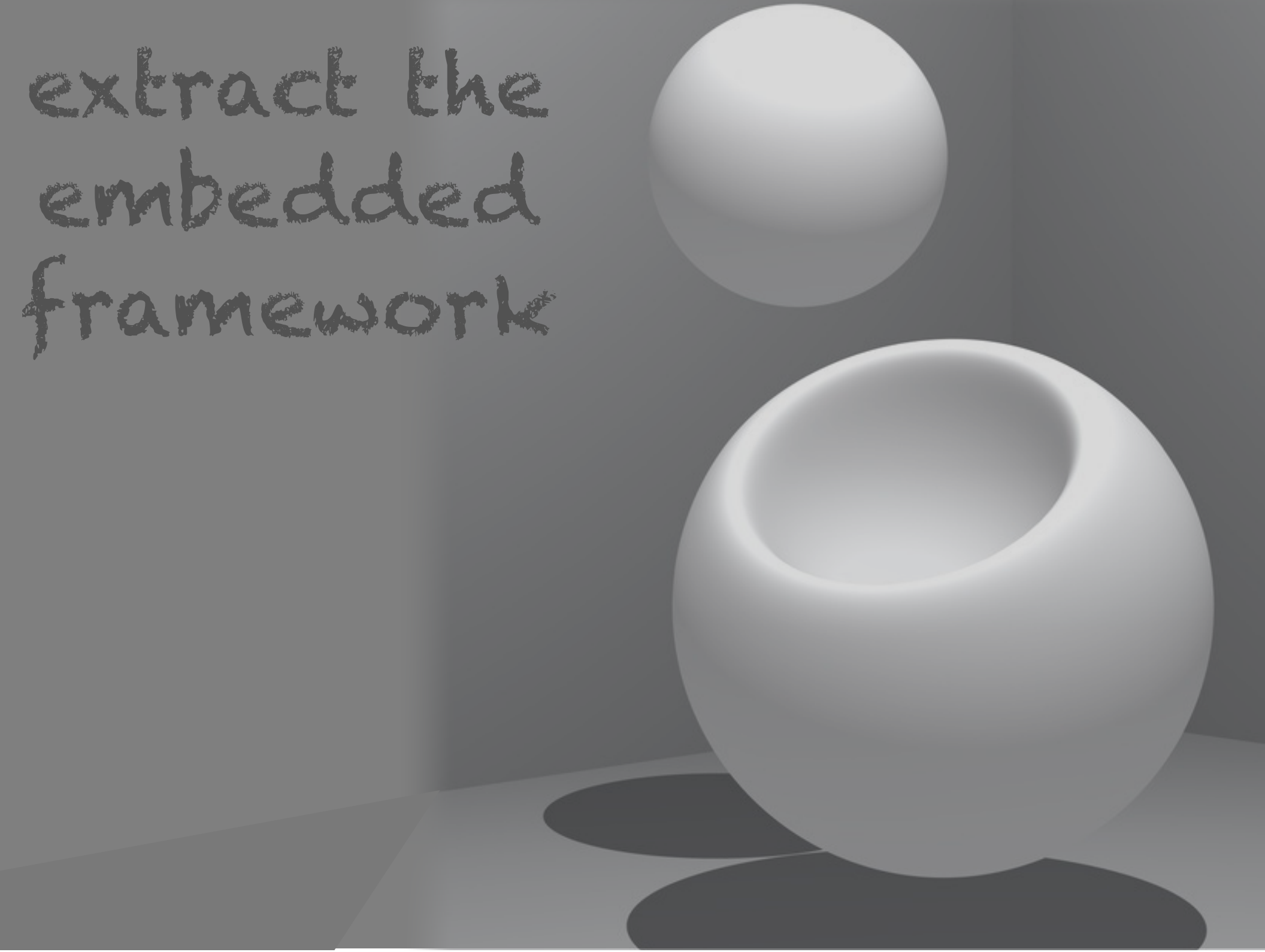
```
public void evaluateParams() {
    addParameter("templateDir", getTemplateDir());
    addParameter("theme", getTheme());

    String name = null;

    if (this.key != null) {

       if(this.name == null) {
            this.name = key;
        }

        if(this.label == null) {
            this.label = "%{getText('"+ key +"')}";
        }

    }

    if (this.name != null) {
        name = findString(this.name);
        addParameter("name", name);
    }

    if (label != null) {
        addParameter("label", findString(label));
    }
```

evaluate.*Params ?

find . -name "*.java" | xargs grep -l "void evaluate.*Params"

./org/apache/struts2/components/AbstractRemoteCallUIBean.java
./org/apache/struts2/components/Anchor.java
./org/apache/struts2/components/Autocompleter.java
./org/apache/struts2/components/Checkbox.java
./org/apache/struts2/components/ComboBox.java
./org/apache/struts2/components/DateTimePicker.java
./org/apache/struts2/components/Div.java
./org/apache/struts2/components/DoubleListUIBean.java
./org/apache/struts2/components/DoubleSelect.java
./org/apache/struts2/components/File.java
./org/apache/struts2/components/Form.java
./org/apache/struts2/components/FormButton.java
./org/apache/struts2/components/Head.java
./org/apache/struts2/components/InputTransferSelect.java
./org/apache/struts2/components/Label.java
./org/apache/struts2/components/ListUIBean.java
./org/apache/struts2/components/OptionTransferSelect.java
./org/apache/struts2/components/Password.java
./org/apache/struts2/components/Reset.java
./org/apache/struts2/components/Select.java
./org/apache/struts2/components/Submit.java
./org/apache/struts2/components/TabbedPanel.java
./org/apache/struts2/components/table/WebTable.java
./org/apache/struts2/components/TextArea.java
./org/apache/struts2/components/TextField.java
./org/apache/struts2/components/Token.java
./org/apache/struts2/components/Tree.java
./org/apache/struts2/components/UIBean.java
./org/apache/struts2/components/UpDownSelect.java

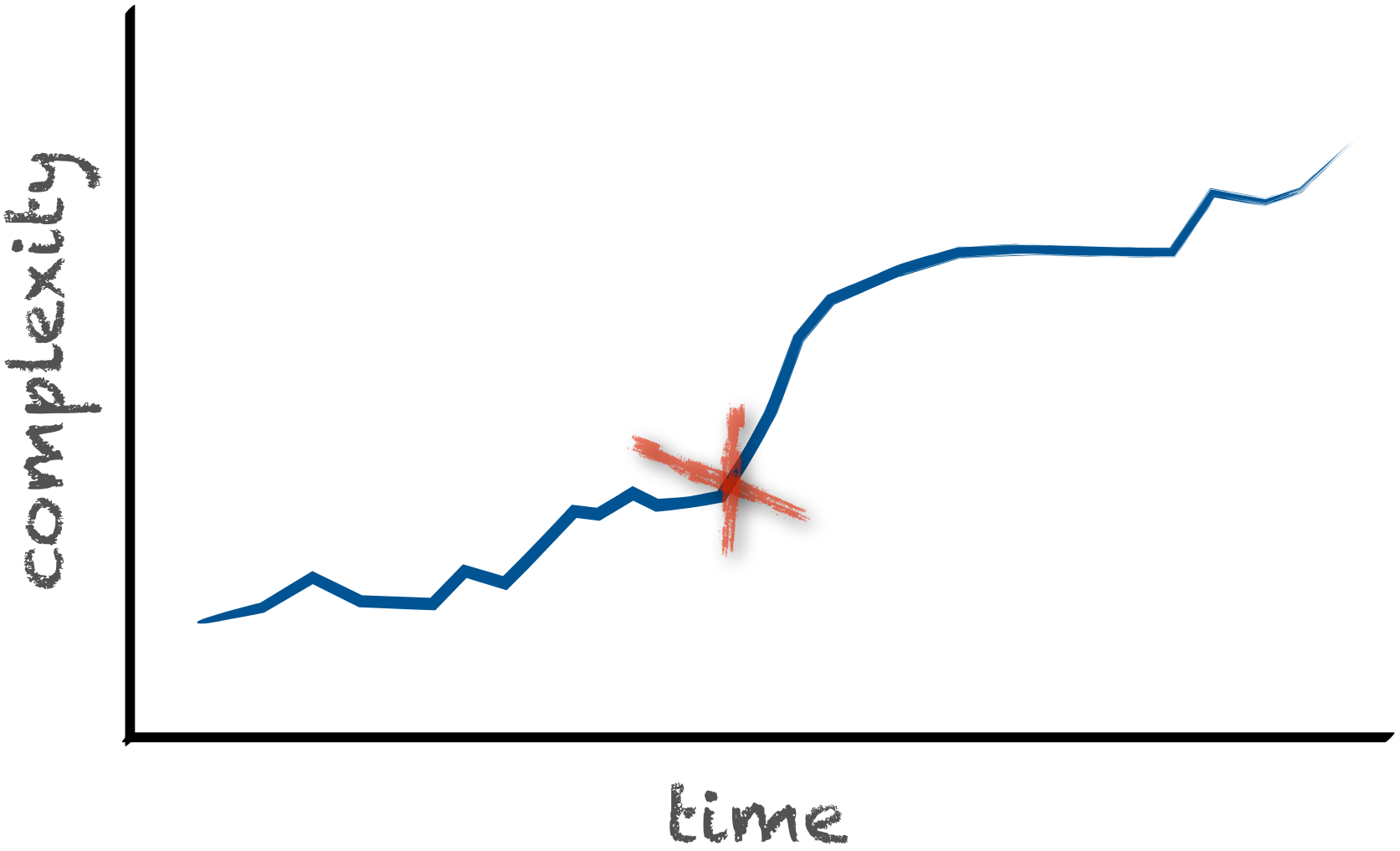
extract the embedded framework



finding & harvesting idiomatic patterns

last responsible moment

last responsible moment




spikes are your
friends

trying to predict the future leads to over-engineering

prefer pro/re-active to predictive

?'s

Mark Richards
Independent Consultant
Hands-on Enterprise / Integration Architect
Published Author / Conference Speaker

http://www.wmrichards.com
http://www.linkedin.com/pub/mark-richards/0/121/5b9

Published Books:
Java Message Service, 2nd Edition
97 Things Every Software Architect Should Know
Java Transaction Design Strategies

Neal Ford
Director / Software Architect /
Meme Wrangler

ThoughtWorks®

2002 Summit Blvd, Level 3, Atlanta, GA 30319, USA
T: +1 40 4242 9929 Twitter: @neal4d
E: nford@thoughtworks.com W: thoughtworks.com