

# Fundamental nano-patterns to characterize and classify Java methods

Jeremy Singer   Gavin Brown   Mikel Luján  
Adam Pocock   Pareskevas Yiapanis

University of Manchester

29 Mar 09

# Outline

1 Overview of Nano-Patterns

2 Nano-Patterns Data Set

3 Conclusions

# What are Nano-Patterns

- Features of Java methods
- ... static (source or bytecode analysis)
- ... simple (for humans and tools)
- ... binary (true or false)

# Example Code

This method exhibits:

- 1 Recursive
- 2 LocalReader
- 3 Exceptions

```
int fact(int n) throws IllegalArgumentException {  
    if (n<0) throw new IllegalArgumentException(n);  
    if (n==0) return 1;  
    else return n*fact(n-1);  
}
```

# Current Nano-Pattern Catalogue

<i>category</i>	<i>name</i>	<i>description</i>
Calling	NoParams NoReturn <b>Recursive</b> SameName <b>Leaf</b>	takes no arguments returns <code>void</code> calls itself recursively calls another method with same name does not issue any method calls
OO	ObjectCreator FieldReader FieldWriter TypeManipulator	creates <code>new</code> objects reads field values from an object writes values to field of an object uses type cast or <code>instanceof</code>
Control Flow	<b>StraightLine</b> Looping Exceptions	no branches in method body loop(s) in method body may throw an unhandled exception
Data Flow	<b>LocalReader</b> LocalWriter <b>ArrayCreator</b> <b>ArrayReader</b> <b>ArrayWriter</b>	reads local var(s) writes local vars creates new array reads values from array writes values to array

# Suggested New Nano-Patterns (more welcome!)

<i>category</i>	<i>name</i>	<i>description</i>
Locality	ThisFieldReader	reads field values from <code>this</code> object
	OtherFieldReader	reads field values from another object
	ThisFieldWriter	writes field values to <code>this</code> object
	OtherFieldWriter	writes field values to another object
	RefsFlowIn	heap references consumed by method
	RefsFlowOut	heap references escape from method
Concurrency	Locker	synchronized data accesses
	Spawner	creates new thread
Polymorphism	Overrider	
	...	...

# Analysis Tool

- Based on ASM bytecode analysis toolkit
- 600 Java source lines of code
- patterns detected by
  - ▶ regexp matches on method signature (NoArgs)
  - ▶ presence of bytecode instructions in method code (ArrayCreator)

# Analysis of Java Benchmarks

program	version	description
Ashes Suite	1st public release	Java compiler test programs
DaCapo	2006-10-MR2	Object-oriented benchmark suite
JBoss	3.2.2	Application server
JEdit	4.3	Java text editor application
JHotDraw	709	Java graphics application
Jikes RVM	2.9.1	Java virtual machine, includes classpath lib
JOlden	initial release	Pointer-intensive benchmark suite
JUnit	4.4	Test harness
SPECjbb	2005	Java business benchmark
SPECjvm	1998	Simple Java client benchmark suite

# Outline

1 Overview of Nano-Patterns

**2 Nano-Patterns Data Set**

3 Conclusions

# Benchmark Corpus Vital Statistics

- 43,880 classes
- 306,531 methods
- For each method, we measure 17 nano-pattern values
- dataset available as gzipped csv file - 3.2MB with method names
- Each method exhibits at least 1 np, mean 4.9 nps

# Nano-Pattern Coverage

nano-pattern	% coverage
LocalReader	89.4
StraightLine	63.6
FieldReader	51.4
Void	50.6
NoParams	39.2
SameName	32.4
LocalWriter	31.1
ObjectCreator	26.5
FieldWriter	26.5
Leaf	20.3
TypeManipulator	15.2
Exceptions	13.6
Looping	11.3
ArrayReader	6.7
ArrayCreator	5.4
ArrayWriter	5.3
Recursive	0.7
Overall	100.0

# Normalized Mutual Information Matrix

(i)

	NP	V	R	SN	L	OC	FR	FW	TM
NP	1.000	0.028	0.016	0.001	0.051	0.007	0.010	0.005	0.025
V	0.028	1.000	0.005	0.043	0.033	0.003	0.032	0.134	0.013
R	0.016	0.005	1.000	0.016	0.038	0.014	0.021	0.000	0.066
SN	0.001	0.043	0.016	1.000	0.181	0.004	0.020	0.009	0.000
L	0.051	0.033	0.038	0.181	1.000	0.141	0.000	0.002	0.053
OC	0.007	0.003	0.014	0.004	0.141	1.000	0.015	0.007	0.037
FR	0.010	0.032	0.021	0.020	0.000	0.015	1.000	0.000	0.046
FW	0.005	0.134	0.000	0.009	0.002	0.007	0.000	1.000	0.005
TM	0.025	0.013	0.066	0.000	0.053	0.037	0.046	0.005	1.000
SL	0.021	0.005	0.139	0.014	0.081	0.085	0.117	0.006	0.130
L	0.014	0.001	0.093	0.003	0.033	0.057	0.054	0.000	0.103
Ex	0.017	0.001	0.005	0.001	0.034	0.046	0.011	0.001	0.006
LR	0.069	0.001	0.019	0.103	0.144	0.001	0.074	0.002	0.041
LW	0.028	0.015	0.118	0.012	0.090	0.110	0.072	0.000	0.178
AC	0.000	0.000	0.002	0.001	0.011	0.041	0.015	0.033	0.020
AR	0.022	0.009	0.035	0.009	0.004	0.022	0.063	0.000	0.025
AW	0.003	0.001	0.006	0.006	0.002	0.031	0.029	0.023	0.016

# Normalized Mutual Information Matrix

(ii)

	SL	L	Ex	LR	LW	AC	AR	AW
NP	0.021	0.014	0.017	0.069	0.028	0.000	0.022	0.003
V	0.005	0.001	0.001	0.001	0.015	0.000	0.009	0.001
R	0.139	0.093	0.005	0.019	0.118	0.002	0.035	0.006
SN	0.014	0.003	0.001	0.103	0.012	0.001	0.009	0.006
L	0.081	0.033	0.034	0.144	0.090	0.011	0.004	0.002
OC	0.085	0.057	0.046	0.001	0.110	0.041	0.022	0.031
FR	0.117	0.054	0.011	0.074	0.072	0.015	0.063	0.029
FW	0.006	0.000	0.001	0.002	0.000	0.033	0.000	0.023
TM	0.130	0.103	0.006	0.041	0.178	0.020	0.025	0.016
SL	1.000	0.361	0.037	0.083	0.331	0.052	0.154	0.078
L	0.361	1.000	0.016	0.039	0.396	0.064	0.244	0.102
Ex	0.037	0.016	1.000	0.002	0.036	0.005	0.002	0.004
LR	0.083	0.039	0.002	1.000	0.118	0.000	0.030	0.000
LW	0.331	0.396	0.036	0.118	1.000	0.101	0.200	0.117
AC	0.052	0.064	0.005	0.000	0.101	1.000	0.079	0.492
AR	0.154	0.244	0.002	0.030	0.200	0.079	1.000	0.146
AW	0.078	0.102	0.004	0.000	0.117	0.492	0.146	1.000

# Association Rule Mining

- Some 'intuitive' rules discovered
- We report three rules in our paper

# Example Rule 1

Looping → TypeManipulator

```
while (i.hasNext()) {  
    Element e = (Element)i.next();  
    // ...  
}
```

# Example Rule 2

## ArrayReader → Looping

```
for (int i=0; i<a.length; i++) {  
    // ...  
    doWork(a[i]);  
    // ...  
}
```

# Example Rule 3

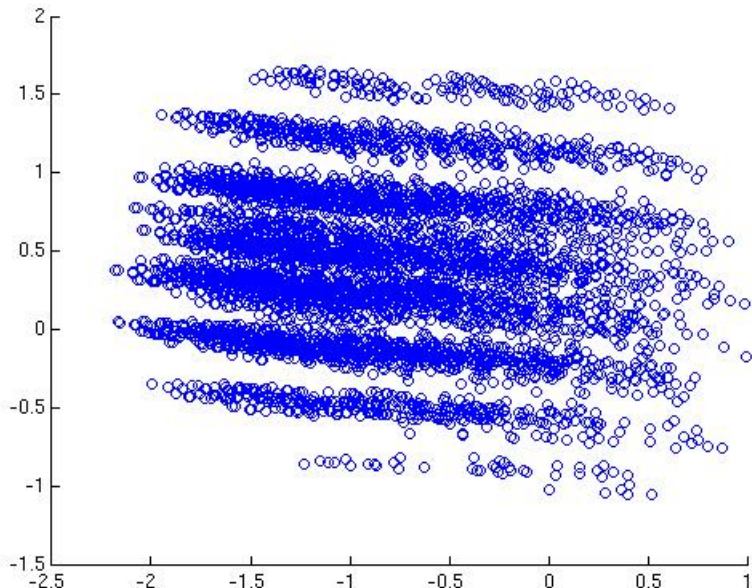
FieldWriter  $\wedge$  StraightLine  $\rightarrow$  NoReturn

```
public void setXYZ(Foo xyz) {  
    this.xyz = xyz;  
    return;  
}
```

# Clustering

- Can we cluster binary data properly?
- Use principal components analysis, reduce to 2d
- Observe sausage-shaped clusters
- requires expert analysis to see if clusters make sense

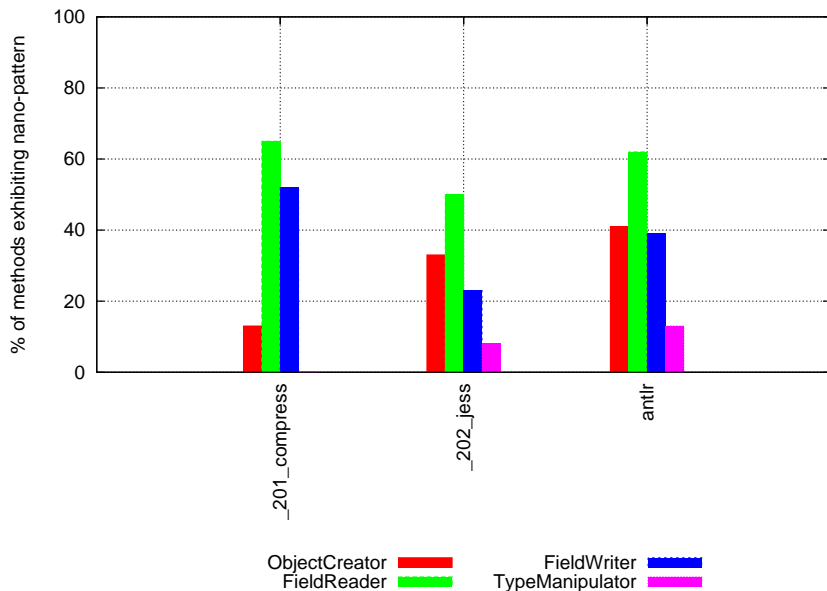
# 2-d Projected Data



# Quantitative Comparison

- Two Java benchmark suites: DaCapo vs SPECjvm98
- Which one is 'better?'
- better = more object-oriented
- we have nano-patterns for object-orientation
- ObjCreator, FieldReader, FieldWriter, TypeManipulator

# Some benchmark profiles



# Turning profiles into individual numbers

- $OO = 0.6OC + 0.1FR + 0.2FW + 0.1TM$
- mean score for DaCapo suite is 29.
- mean for SPECjvm98 suite is 28.
- too close to call!

# Outline

1 Overview of Nano-Patterns

2 Nano-Patterns Data Set

**3 Conclusions**

# Nano-Patterns are useful!

- concise characterization of Java methods
- useful for
  - ▶ quantitative analysis
  - ▶ machine learning
- work in progress . . .