FUTURE CHALLENGES IN SOFTWARE EVOLUTION ANALYSIS FROM INDUSTRIAL & ACADEMIC PERSPECTIVES

Aiko Yamashita

Oslo and Akershus University College of Applied Sciences

2016-07-12
Career trajectory...
High-level research focus/question:

How can we use different** metrics to attain better maintainability assessments and to support software evolution-related activities?

**product-related and project-related (empirical) metrics
METRICS FOCUS: **CODE SMELLS**

A hint about suboptimal implementation choices that can affect negatively future maintenance and evolution.
METRICS FOCUS: CODE SMELLS

Shotgun Surgery
A change leads to another change, to another, to another.

Move method refactoring
Reduce the coupling between components
METRICS FOCUS: **CODE SMELLS**

- **Refactoring ROI not clear**: e.g., to eliminate a code smell implies a cost (refactoring, rework) and a risk (introduction defects).

- Not clear in which **contexts** (e.g., activities) smell-based analysis **performs best**, and which are the **preconditions** (e.g., additional data) required.

**State of art**

- Tooling: JDeodorant
- Detection: InFusion/InCode
- Analysis: Analyze4J
- PTIDEJ / DECOR
- NDepend

**Challenges**
Goal: Assess whether code smells can be used effectively for assessing the maintainability of software.

Research method: Longitudinal, in-vivo case study investigating a Maintenance Project involving 4 Java systems and 6 software professionals.

Research techniques: case replication, cross-case synthesis, explanatory models (e.g., regression), grounded theory.
EMPIRICAL STUDY

Context

- 4 Java Applications
- Same functionality
- Different design/code
- Size: 7KLOC to 14KLOC

Study Design

- System: A, B, C, D
- Developer

Maintenance Tasks

- Task 1. Replacing external data source
- Task 2. New authentication mechanism
- Task 3. New Reporting functionality
INSIDE THE BELLY OF THE MONSTER

** Variables of interest **
- Code smells (num. smells**, smell density**)  
- Open interviews Audio files/notes
- Study diary
- Daily interviews Audio files/notes
- Trac (Issue tracker), Acceptance test reports

** Data sources **
- Source code
- Eclipse activity logs
- Think aloud Video files/notes
- Task progress sheets

** Project context **
- System
- Programming Skill
- Development Technology

** Tasks **
- System
- Tasks
- Programming Skill
- Development Technology

** Moderator variables **
- Variables of interest
- Data sources

** Maintenance outcomes **
- Maintainability perception*
- Maintenance problems**
- Change Size**
- Effort**
- Defects*

** Fact-sheet **
- 50,000 Euros
- Sep-Dec, 2008
- 7 Weeks
- 6 Developers
- 2 Companies

** Variables of interest **
- Code smells (num. smells**, smell density**)  
- Open interviews Audio files/notes
- Study diary
- Daily interviews Audio files/notes
- Trac (Issue tracker), Acceptance test reports

** Data sources **
- Source code
- Eclipse activity logs
- Think aloud Video files/notes
- Task progress sheets

** System and file level **
* Only at system level

I'd like some fresh, clear, well-seasoned perspective.

Anton Ego
LET’S GET SOME “PERSPECTIVES”
LET’S GET SOME “PERSPECTIVES”
Can code smells Indicate system-level maintainability?

**Maintainability Assessment**

- Systems were ranked according to their no. of code smells, and their smell density (no. smells/KLOC).

**Actual Maintainability**

- Systems were ranked according to their maintainability, which was measured by: effort (time) and defects introduced.

- Standardized scores were calculated for the ranking

**Do they correspond?**

*In addition, smell-based assessment was compared to two previous assessments (CK metrics and Expert judgment) on the systems*
Can code smells indicate system-level maintainability?
Can code smells indicate system-level maintainability?
Can code smells Indicate system-level maintainability?

Number of code smells displayed highest correspondence to actual maintainability

<table>
<thead>
<tr>
<th>Evaluation approaches</th>
<th>Level of matching or agreement</th>
<th>Kappa coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Measures</td>
<td>Matching D as most maintainable and A as intermediate.</td>
<td>0.50</td>
</tr>
<tr>
<td>Expert Judgment</td>
<td>Matching D as most maintainable and B as least maintainable.</td>
<td>0.50</td>
</tr>
<tr>
<td>Number smells</td>
<td>Matching C as most maintainable, A as intermediate and B as least maintainable.</td>
<td>0.75</td>
</tr>
<tr>
<td>Smell density</td>
<td>No matching with maintenance outcomes</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**Expert Judgment** was considered as the most flexible approach, because it considers both the effects of **system size** and potential **maintenance scenarios**
Do code smell cover important maintainability attributes?

- Appropriate technical platform
- Coherent naming
- Design suited to the problem domain
- Initial defects
- Architecture

Not covered:

- Encapsulation
- Inheritance Libraries
- Simplicity
- Use of components
- Design consistency
- Logic Spread

Partially covered:

- Duplicated code

Covered:

A Yamashita, L Moonen. Do code smells reflect important maintainability aspects?
LET’S GET SOME “PERSPECTIVES”
Can code smells explain maintenance effort or problems?

**Explanatory model for Problem**

**Dependent variable:** Problematic?

**Independent variables:** 12 smells

**Control variables:**
- File size (LOC)
- Churn
- System

**Analysis:** Logistic Regression Model

**Explanatory model for Effort**

**Dependent variable:** Effort (time)

**Independent variables:** 12 smells

**Control variables:**
- File size (LOC)
- Number of revisions on a file
- System
- Developer
- Round

**Analysis:** Multiple Regression Model

In addition, principal component analysis (PCA) on the code smell distribution and qualitative analysis was performed.
Can code smells explain maintenance effort or problems?

Explanatory model for Problem

- Interface Segregation Principle Violation (ISPV) was able to explain problems \([\text{Exp}(B) = 7.610, p = 0.032]\]
- Data Clump significant contributor of model \([\text{Exp}(B) = 0.053, p = 0.029]\) but associated to less problems!
- PCA indicated that ISPV tends to not be associated to code smells that are related to size.
- Qualitative data suggests that ISPV is related to error/change propagation, and difficult concept location.
Can code smells explain maintenance effort or problems?

Explanatory model for Problem

- Interface Segregation Principle Violation (ISPV) was able to explain problems \[ \text{Exp}(B) = 7.610, \ p = 0.032 \]
- Data Clump significant contributor of model \[ \text{Exp}(B) = 0.053, \ p = 0.029 \] but associated to less problems!
- PCA indicated that ISPV tends to not be associated to code smells that are related to size.
- Qualitative data suggests that ISPV is related to error/change propagation, and difficult concept location.

Some code smells can potentially explain the occurrence of problems during maintenance. Also, not all smells seem to be problematic…
Can code smells explain maintenance effort or problems?

**Explanatory model for Problem**

- Interface Segregation Principle Violation (ISPV) was able to explain problems ($\text{Exp(B)} = 7.610$, $p = 0.032$).
- Data Clump was a significant contributor of the model ($\text{Exp(B)} = 0.053$, $p = 0.029$), but associated to less problems!
- PCA indicated that ISPV tends to not be associated to code smells that are related to size.
- Qualitative data suggests that ISPV is related to error/change propagation, and difficult concept location.

**Explanatory model for Effort**

- A model that includes file size and number of changes and code smells displayed a fit of $R^2 = 0.58$.
- Removing the smells from that model did not decrease the fit! ($R^2 = 0.58$)
- Only smell that remained significant was Refused Bequest, which registered a **decrease** in effort ($\alpha < 0.01$).
- File size and number of changes remain the most significant predictors of effort ($\alpha < 0.001$).

Some code smells can potentially explain the occurrence of problems during maintenance. Also, not all smells seem to be problematic…
Can code smells explain maintenance effort or problems?

**Explanatory model for Problem**

- Interface Segregation Principle Violation (ISPV) was able to explain problems \[\text{Exp}(B) = 7.610, p = 0.032\]
- Data Clump significant contributor of model \[\text{Exp}(B) = 0.053, p = 0.029\] but associated to less problems!
- PCA indicated that ISPV tends to not be associated to code smells that are related to size.
- Qualitative data suggests that ISPV is related to error/change propagation, and difficult concept location.

**Explanatory model for Effort**

- A model that includes file size and number of changes and code smells displayed a fit of \(R^2 = 0.58\)
- Removing the smells from that model did not decrease the fit!! \(R^2 = 0.58\)
- Only smell that remained significant was Refused Bequest, which registered a decrease in effort \(\alpha < 0.01\)
- File size and number of changes remain the most significant predictors of effort \(\alpha < 0.001\)

Some code smells can potentially explain the occurrence of problems during maintenance. Also, not all smells seem to be problematic…

Code smells are not better at explaining sheer-effort at file level, than size and number of revisions.
LET’S GET SOME “PERSPECTIVES”
To what extent can code smells explain maintenance problems?

Analysis

Observational study
Daily interviews
Think-aloud protocol

Maintenance Difficulties

% Non-Source code related difficulties

% Source code related difficulties

% Code-smell related difficulties

% Non-code-smell related difficulties

Principal Component Analysis

Dependency Analysis
To what extent can code smells explain maintenance problems?

Distribution of maintenance problems according to source:

- **Runtime environment (10%)**
- **Initial defects (3%)**
- **Unclear specs (1%)**
- **Dep. on external services (2%)**
- **Code smells (27%)**
- **Architecture (4%)**
- **Inadequate infrastructure (23%)**
- **Developer's strategy (9%)**
- **Combined factors (6%)**
- **Other code characteristics (14%)**

To what extent can maintenance problems be predicted by code smell detection?—An empirical study

A Yamashita, L Moonen
How do Code Smells interact with one another?

Some patterns where identified:

- **Hoarders**
  - Feature Envy
  - God Class
  - God Method

- **Data Containers**
  - Data Clump
  - Data Class

- **Wide Interfaces**
  - ISP Violation
  - Shotgun Surgery

- **Confounders**
  - Temporal variable used for several purposes
  - Duplicated code in conditional branches

Exploring the impact of inter-smell relations on software maintainability: An empirical study
A Yamashita, L Moonen
The major reason why the developers found System A difficult to understand seems to be due to the proportion of afferent coupling. Also, they do not seem to relate much to the size of the code (Factor 1). This characteristic may, consequently, be considered to the project. How do Code Smells interact with one another?

### Table

<table>
<thead>
<tr>
<th>File</th>
<th>System</th>
<th>DC</th>
<th>CL</th>
<th>DUP</th>
<th>FE</th>
<th>GC</th>
<th>GM</th>
<th>ISPV</th>
<th>MC</th>
<th>RB</th>
<th>SS</th>
<th>Temp</th>
<th>Imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudyDatabase</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PrivilegesManageAction</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>StudiesEditAction</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StudiesSearchAction</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StudySearch</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DB</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StudyDAO</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table

<table>
<thead>
<tr>
<th>File</th>
<th>Individual code smells</th>
<th>Coupled smells</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudySearch.java</td>
<td>GC, GM, FE</td>
<td>FE, GM, ISPV, GC, SS</td>
</tr>
<tr>
<td>ObjectStatementImpl.java</td>
<td>ISPV, SS</td>
<td></td>
</tr>
<tr>
<td>MemoryCachingSimula.java</td>
<td>GC, TMP</td>
<td>ISPV, GC, SS, TMP</td>
</tr>
<tr>
<td>Simula.java</td>
<td>ISPV, SS</td>
<td></td>
</tr>
</tbody>
</table>
How do Code Smells interact with one another?

Coupled smells can have similar implications as collocated smells!

<table>
<thead>
<tr>
<th>File</th>
<th>Individual code smells</th>
<th>Coupled smells</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudySearch.java</td>
<td>GC, GM, FE</td>
<td>FE, GM, ISPV, GC, SS</td>
</tr>
<tr>
<td>ObjectStatementImpl.java</td>
<td>ISPV, SS</td>
<td></td>
</tr>
<tr>
<td>MemoryCachingSimula.java</td>
<td>GC, TMP</td>
<td>ISPV, GC, SS, TMP</td>
</tr>
<tr>
<td>Simula.java</td>
<td>ISPV, SS</td>
<td></td>
</tr>
</tbody>
</table>

Problematic files with at least one God Method

System B

A

God Class
Feature Envy

≈

B

God Class
Coupling

Feature Envy
An interesting example case for the interaction effect between code smells and other design flaws…

**Typed getters and setters**
OVERALL FINDINGS

• **Aggregated** and **individual** code smell analyses are **insufficient** to understand the role of code smells on maintenance.

• **Code smells interact!** (collocated and coupled code smells)

• An approach (more promising?) is to **incorporate dependency analysis** to the study of individual **code smells**.

• There may be **other smells not yet discovered**...

• Role of **code smells** are **dependent** of the maintenance **context** (ex. Data Clumps)
FOLLOW-UP STUDIES?

...LET'S GO EXPLORING!
Can we find the same inter-smells in larger systems?

**Open Source Systems**

**Name:** ElasticSearch  
**Java:** 2951 files  
**Total Size:** 253 KLOC  
**History:** 102 minor releases and 22 major releases since 2010

**Name:** Mahout  
**Java:** 935  
**Scala:** 12  
**Total Size:** 92 KLOC  
**History:** 10 releases since 2010

**Industrial System**

**Name:** Ebehandling  
**Java:** 5300 files  
**Total:** 5840 files  
**Total Size:** 601 KLOC  
**History:** 40 major releases and 15 patch releases since 2009

**Name:** EbeanSearch  
**Java:** 2951 files  
**Total Size:** 253 KLOC  
**History:** 102 minor releases and 22 major releases since 2010

A Yamashita, M Zanoni, FA Fontana, B Walter. Inter-smell relations in industrial and open source systems: A replication and comparative analysis
IGNORING COUPLED SMELLS INCREASES CHANCES OF INTER-SMELL FALSE NEGATIVES

<table>
<thead>
<tr>
<th>System (Component)</th>
<th>Smells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ebehandling (9)</td>
<td>Data Class</td>
</tr>
<tr>
<td>Mahout (10)</td>
<td>Feature Envy</td>
</tr>
<tr>
<td>ElasticSearch (8)</td>
<td></td>
</tr>
<tr>
<td>Ebehandling (7)</td>
<td>Sibling Duplication</td>
</tr>
<tr>
<td>ElasticSearch (14)</td>
<td>Message Chains</td>
</tr>
<tr>
<td>Mahout (5)</td>
<td>God Class</td>
</tr>
<tr>
<td>ElasticSearch (1)</td>
<td>Schizo Class</td>
</tr>
<tr>
<td>Ebehandling (1)</td>
<td>God Class</td>
</tr>
<tr>
<td>Mahout (5)</td>
<td>Data Class</td>
</tr>
<tr>
<td>ElasticSearch (7)</td>
<td>Message Chains</td>
</tr>
<tr>
<td>Mahout (9)</td>
<td>God Class</td>
</tr>
<tr>
<td>ElasticSearch (7)</td>
<td>Feature Envy</td>
</tr>
<tr>
<td>Ebehandling (4)</td>
<td>Feature Envy</td>
</tr>
<tr>
<td>ElasticSearch (13)</td>
<td>Message Chains</td>
</tr>
<tr>
<td>Ebehandling (5)</td>
<td>Internal Duplication</td>
</tr>
<tr>
<td>Mahout (2)</td>
<td>External Duplication</td>
</tr>
</tbody>
</table>

**Pattern 1**
- Feature Envy → Data Class
- Ebehandling (9), Mahout(10)

**Pattern 2**
- Data Class → Class
- Ebehandling (2), ElasticSearch (3)

**Pattern 3**
- Message Chains → Sibling Duplication
- Ebehandling (7), ElasticSearch (14)

**Pattern 4**
- Data Class → Feature Envy
- ElasticSearch (8)

**Pattern 5**
- Feature Envy → Class
- Ebehandling (1), ElasticSearch (1)

**Pattern 6**
- Message Chains → Sibling Duplication
- Ebehandling (2), ElasticSearch (3)

**Pattern 7**
- Internal Duplication → Class
- Mahout (2)
- External Duplication → Class
- Mahout (2)
- Class → External Duplication
- Mahout (2)
- Internal Duplication → Class
- Ebehandling (6)
INDUSTRIAL SYSTEMS ARE DIFFERENT FROM OPEN SOURCE

(a) Mahout

(b) Ebehandling

(c) ElasticSearch

Data Clumps having dependencies on other Data Clumps

Pattern 9

Data Clumps 
Ebehandling (8), Mahout(6), ElasticSearch(5)
Classes having incoming dependencies from God Classes and Feature Envy methods

INDUSTRIAL SYSTEMS ARE DIFFERENT FROM OPEN SOURCE
How do Code Smells affect Maintenance Activities?

-MINING IDE EVENT LOGS-

Reading
Programming (code-related) activities during Maintenance
Navigating
Searching
Editing
Others

MINING EVENT LOGS TO UNDERSTAND EFFORT

- Selection of artifacts in the package explorer
- Selection of Java elements in the editor window
- Selecting Java elements in the file outline
- Editing source files (Java files)
- Scrolling the source code window
- Switching between open files
- Running Eclipse “commands” (copy, paste, go to line)

Activity logs
DISTRIBUTION OF ACTIVITY EFFORT

- Mostly performed activities: Navigating (58.72%), Reading (28.27%), Editing (10.18%) and searching (2.47%)

- Distribution is consistent with Ko et al. 2006 (top four)

- Reading as most consuming activity in Ko et al. 2006.
  - Definition of event/action belonging to an activity

For our analysis, we only consider: Editing, Navigating, Searching and Reading
CODE SMELLS AFFECT SOME TASKS...

Maintenance problems in previous work related to increased effort for editing, navigating and reading

<table>
<thead>
<tr>
<th>Code Smell</th>
<th>Editing</th>
<th>Navigating</th>
<th>Reading</th>
<th>Searching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Clump</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicated Code in conditional branches</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature Envy</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>God Class</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>God Method</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISP Violation</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Misplaced Class</td>
<td>-</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Refused Bequest</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shotgun Surgery</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temporary variable is used for several purposes
Use interface instead of implementation

“+”: require more effort
“−”: required less effort
“empty”: no effect on the effort

Smells explain better Editing and Navigating effort than file size, but not for Reading and Searching
Research questions:

• What are refactoring tendencies?
• Individual refactoring styles?
• Learning when refactoring?

• Do design or type of task affect refactoring?
Only for developer 3, there was an effect from system, we suspect due to a more “ad-hoc” refactoring strategy.

---

**INDIVIDUAL STYLE FOR REFACTORING?**

---

**Paired Differences**

<table>
<thead>
<tr>
<th>Pair</th>
<th>Dev1-SysA - Dev1-SysB</th>
<th>Dev1-SysB - Dev2-SysA</th>
<th>Dev2-SysA - Dev3-SysC</th>
<th>Dev3-SysC - Dev3-SysD</th>
<th>Dev3-SysD - Dev4-SysC</th>
<th>Dev4-SysC - Dev5-SysB</th>
<th>Dev5-SysB - Dev6-SysA</th>
<th>Dev6-SysA - Dev6-SysB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>-1.143</td>
<td>.238</td>
<td>.333</td>
<td>.143</td>
<td>.286</td>
<td>.238</td>
<td>.768</td>
<td>.619</td>
</tr>
<tr>
<td>St.Dev</td>
<td>11.800</td>
<td>2.189</td>
<td>.730</td>
<td>.727</td>
<td>5.640</td>
<td>7.680</td>
<td>1.686</td>
<td>1.168</td>
</tr>
<tr>
<td>Std. Error</td>
<td>Mean</td>
<td>2.575</td>
<td>.478</td>
<td>.159</td>
<td>.159</td>
<td>.159</td>
<td>.159</td>
<td>.159</td>
</tr>
<tr>
<td>95% Confidence Interval of the Mean</td>
<td>-6.514, 4.228</td>
<td>-.758, 1.124</td>
<td>.001, .666</td>
<td>-.186, .474</td>
<td>-.228, 2.853</td>
<td>.588, 1.420</td>
<td>.049, .711</td>
<td>.171, .819</td>
</tr>
<tr>
<td>t</td>
<td>.444</td>
<td>.499</td>
<td>.666</td>
<td>2.092</td>
<td>.900</td>
<td>.232</td>
<td>.588</td>
<td>.171</td>
</tr>
<tr>
<td>df</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Sig. (2-tailed)</td>
<td>.662</td>
<td>.624</td>
<td>.049</td>
<td>.379</td>
<td>.619</td>
<td>.191</td>
<td>.049</td>
<td>.171</td>
</tr>
</tbody>
</table>
Only for developer 3, there was an effect from system, we suspect due to a more “ad-hoc” refactoring strategy.
STILL.. EFFECTS FROM TASK TYPE AND CODE SMELLS WERE MINOR...

Only few code smells could explain some refactorings (some with negative coefficients)

For tasks, it was much more the task size what could explain the frequency, rather than the type of task

<table>
<thead>
<tr>
<th>Model</th>
<th>-2 Log likelihood</th>
<th>Cox &amp; Snell R Square</th>
<th>Nagelkerke R Square</th>
<th>Variables contributing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change method signature</td>
<td>72,718</td>
<td>.081</td>
<td>.371</td>
<td>Data Clump (Beta = -4.093 p &lt; 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data Class (Beta = 5.568 p &lt; 0.05)</td>
</tr>
<tr>
<td>Organize imports</td>
<td>473,871</td>
<td>.102</td>
<td>.149</td>
<td>Feature Envy (Beta = 4.303 p &lt; 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Shotgun Surgery (Beta = -3.657 p &lt; 0.001)</td>
</tr>
<tr>
<td>Extract method</td>
<td>256,678</td>
<td>.103</td>
<td>.209</td>
<td>Total no. smells (Beta = -3.414 p&lt;0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Lines of code (Beta = 0.197 p &lt; 0.01)</td>
</tr>
</tbody>
</table>
EFFECTS OF REFACTORIZING ON CODE EVOLUTION

(results from a series of binary logistic regression models)

It appears as extract method could increase odds of defects!!
WE ARE DEALING WITH AN EVEN MORE MESSIER PROBLEM (THAN WE THOUGHT)
WE ARE DEALING WITH AN EVEN MORE MESSIER PROBLEM (THAN WE THOUGHT)

CONTEXTUAL VARIABILITY
(ACTIVITY TYPES, FLOSS VS. IND)

UNCLEAR CATEGORIZATION AND EFFECTS FROM TASKS

ARTIFICIAL (USELESS) FORMALISMS?
Historical adaptable data-mining capability for sw evolution

Inductive approach to identification of harmful design/code attributes

- Adaptable mining-based capability that can “read” the situation based on code evolution, defects and behavioural patterns from developers
- Identification of problematic cases via think-alouds
- Event patterns can help identifying problematic artifacts
- Examination of artifacts can hopefully lead to measurable attributes on the artifacts
Adaptable mining-based capability that can “read” the situation based on code evolution, defects and behavioural patterns from developers.

Identification of problematic cases via think-alouds.

Event patterns can help identifying problematic artifacts.

Examination of artifacts can hopefully lead to measurable attributes on the artifacts.
Adaptable mining-based capability that can “read” the situation based on code evolution, defects and behavioural patterns from developers

Identification of problematic cases via think-alouds

Event patterns can help identifying problematic artifacts

Examination of artifacts can hopefully lead to measurable attributes on the artifacts
BUT... HOW DO WE GO ABOUT INDUCTIVE ANALYSIS?

**Challenge** (1 out of N): Badly integrated empirical data

- Routines/tooling are not integrated into the work-flow...
- No culture for registering the underlying cause of code change...
- No "reflective routines" powered by toolsets that are easy to use...
IDEA: TOOLSET AND ROUTINES FOR SEAMLESS INTEGRATION OF CODE AND REPOSITORY ANALYSIS

1. Technology should be available

2. Which can be tested in case studies via feasible routines

BUT better "packaging" is needed!

**INTEGRATION OF CODE AND REPOSITORY ANALYSIS**

**IDEA:** TOOLSET AND ROUTINES FOR SEAMLESS INTEGRATION OF CODE AND REPOSITORY ANALYSIS

**JIRA**

**MetricsGrimoire +**

**git**

**BUT** better "packaging" is needed!

1. Technology should be available

2. Which can be tested in case studies via feasible routines

**IDEA:** TOOLSET AND ROUTINES FOR SEAMLESS INTEGRATION OF CODE AND REPOSITORY ANALYSIS

**JIRA**

**MetricsGrimoire +**

**git**

**BUT** better "packaging" is needed!
Are we even concerned about the same smells?

EXPLORATORY SURVEY

Where do you read about code smells?

- Colleagues, seminars (10)
- Forums and wikipedia (6)
- Tool Vendors' websites (5)
- Scientific Papers (5)
- Books (18)
- Gurus' websites (9)
- Internet Forums (32)
- Blogs (38)
FRAGMENTATION IS REAL

Cross and Hagardon. “Critical Connections: Driving Rapid Innovation with a Network Perspective”
Know-what

Know-how

Implications!

Our system of values...

A. Yamashita. Integration of SE Research and Industry: Reflections, Theories and Illustrative Example
How bad is smelly code?

—Measuring the effect of code smells

By Aiko Yamashita and Thomas Flemming

Running scientific experiments to figure out what can make code easy to understand, use and maintain.

What to do with code that smells?

We have all been taught that our main responsibility as programmers is to write code that is easy to understand. But what happens when you are a lonely programmer writing a few lines of Python to add a feature to a system that you do not understand? The important thing is that the code you write is easy to understand by your future self, and by other developers who will maintain the code through the life span of the software product, often saves you and your colleagues many hours of Figure 9: A part of the result of the executed code.
“If you don't make it to air, there is nothingness. You're dead.”

- Jack Welch
Prejudices on results coming from academia:
“too complicated”,”too theoretical”,”boring”, “not as cool as..”
Wave of technology adoption may obey a different set of rules than our current understanding...
Maybe we need to understand better the phenomenon of adoption from a sociological perspective (e.g., perception).
There has been an ‘evolution’ on FLOSS “perception”

Concept of Sand-box

Platform support, Tutorials, etc

Now many OSS projects are “sleek”... and why not?
BUT... HOW DO WE GO ABOUT INDUCTIVE ANALYSIS?

Challenge (2 out of N): No clear classification schemas

- No clear/useful/easy to use taxonomy for classifying faults..
- No clear taxonomy for classifying underlying reasons for changes
- Unclear description of contexts/domains of given projects (comparability may be a challenge, even if the data is correctly classified)
IDEA: TAXONOMY FOR DEFECT CATEGORIZATION THAT IS EASY TO USE BY DEVELOPERS
THE FALSE POSITIVES PROBLEM

do I really need to check all of them?
IDEA: TAXONOMY FOR DESCRIBING PROJECT CONTEXTS AND DOMAINS (ANOTHER “GRAND” IDEA)

Mainly for generating more adaptable quality models based on empirical data from given project repository

Machine learning + Crowd sourcing
IDEA: TAXONOMY FOR DESCRIBING PROJECT CONTEXTS AND DOMAINS (ANOTHER “GRAND” IDEA)

Mainly for generating more adaptable quality models based on empirical data from given project repository.

Via crowdsourcing, we ask people what domain a project is and we train a model to guess the domain when analyzing projects in GitHub.
BUT... HOW DO WE GO ABOUT INDUCTIVE ANALYSIS?

**Challenge** (3 out of N): No (empirical) data available

Only millennial files of code after code, complemented with some other artefacts... (like outdated documentation)
IDEA: USE OF BENCHMARKS, THRESHOLD ANALYSIS (AND OTHERS...)

Concept has been proposed by Alves, Ferreira, Baggen, Oliveira...

REST API

Benchmark (Set of .csv files)

Cloned and analyzed

A Yamashita. Experiences from Performing Software Quality Evaluations via Combining Benchmark-Based Metrics Analysis, Software Visualization, and Expert Assessment
IDEA: USE OF BENCHMARKS, THRESHOLD ANALYSIS (AND OTHERS...)

A concept has been proposed by Alves, Ferreira, Baggen, Oliveira...

<table>
<thead>
<tr>
<th>Language</th>
<th>Repository</th>
<th>Sample Size</th>
<th>Min LOC</th>
<th>Max LOC</th>
<th>Median LOC</th>
<th>Mean LOC</th>
<th>SD dev. LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language 1</td>
<td>Github</td>
<td>106 090</td>
<td>1 048</td>
<td>24 308</td>
<td>21 830</td>
<td>50 784</td>
<td></td>
</tr>
<tr>
<td>Language 2</td>
<td>Github</td>
<td>8 559</td>
<td>1 044</td>
<td>26 149</td>
<td>73 927</td>
<td>65 094</td>
<td></td>
</tr>
<tr>
<td>Language 3</td>
<td>Source forge</td>
<td>167</td>
<td>3 109</td>
<td>9 953</td>
<td>18 232</td>
<td>22 564</td>
<td></td>
</tr>
</tbody>
</table>

HOW RELIABLE IS THE BENCHMARK?
The discipline of software engineering has grown in maturity and rigor over the past twenty years. One of the goals of software engineering research is to achieve generality, which is not easy. The findings of a study about a population are considered as general (and thus research has importance) if the technique being evaluated on projects other than those in the sample yield similar results.

While it is common sense to select a sample that is representative of a population, the importance of diversity and representativeness、“proportional” are orthogonal concepts. A perfectly representative sample would be 4×X and 1×Y. In this case, 100 subjects of type X and 1×Y subgroup in the population. In the example above, diversity (“roughly equal size”) is to ensure that studies are developed guidelines to improve diversity and representativeness. The aim of such guidelines is to ensure that studies are monitored by Ohloh.net of 20,000 active open source projects, and to increase the coverage the given sample. We introduce a measure called sample coverage, defined as the percentage of projects in the sample, and to select the projects that increase the coverage the.

Improve the quality of the sample size does not necessarily better.

Diversity in Software Engineering Research

Meiyappan Nagappan
Software Analysis and Intelligence Lab
Queen’s University, Kingston, Canada
mei@cs.queensu.ca

Thomas Zimmermann
Microsoft Research
Redmond, WA, USA
tzimmer@microsoft.com

Christian Bird
Microsoft Research
Redmond, WA, USA
Christian.Bird@microsoft.com

ABSTRACT
One of the goals of software engineering research is to achieve generality. Are the phenomena found in a few projects reflective of others? Will a technique perform as well on projects other than the projects it is evaluated on? While it is common sense to select a sample that is representative of a population, the importance of diversity is often overlooked, yet as important. In this paper, we combine ideas from representativeness and diversity and introduce a measure called sample coverage, defined as the percentage of projects in a population that are similar to the given sample. We introduce algorithms to compute the sample coverage for a given set of projects and to select the projects that increase the coverage the.

et al. [2] examined 1,000 projects. Another example is the study by Gabel and Su that examined 6,000 projects [3]. But if care isn’t taken when selecting which projects to analyze, then increasing the sample size does not actually contribute to the goal of increased generality. More is not necessarily better.

As an example, consider a researcher who wants to investigate a hypothesis about say distributed development on a large number of projects in an effort to demonstrate generality. The researcher goes to the json.org website and randomly selects twenty projects, all of them JSON parsers. Because of the narrow range of functionality of the projects in the sample, any findings will not be very representa...
LAST (BUT NOT LEAST) IDEA: MORE FOCUS ON DEVELOPERS?

Software Engineering Researchers

Software Engineers

Smell detector

Hey! Get back to work!

Compiling!

Oh. Carry on.

This is the picture we want to avoid...
What’s cooking?
Social Network as Collaboration Market

Collaboration Needs/Interests
Expertise/Domain Areas
Skill Sets
Knowledge and Results
Available Tools
Collaboration Schemas
Time availability and Resources
Geographical Location
Interest Areas/Affinities
Organisational/National Culture

Technology exists, but the **domain** needs to be modelled... Working on interviews and surveys :)

**Matching criteria**
“We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win.”

- John Kennedy
ありがとうございます!
(thank you!)

Contact: aiko.fallas@gmail.com