Empirical Analysis of Abnormalities in Local Variables of Change-Prone Java Methods

Hirohisa Aman¹)
Sousuke Amasaki²)
Tomoyuki Yokogawa²)
Minoru Kawahara¹)

¹) Ehime University, Japan
²) Okayama Prefectural University, Japan
Overview

Purpose
To reveal relationships of local variables to code quality; We will analyze their names, types and scopes

Method
◦ Surveyed the real trend of local variable's name, type and scope in six popular open source Java programs
◦ Then, evaluated an abnormality of local variable by using the Mahalanobis distance

Result
◦ Methods having an abnormal variable are about 1.2-2.5 times more likely to be change-prone, etc.
Outline

- Background & Motivation
- Properties of Local Variables
- Abnormality of Local Variables
- Empirical Study
  - Aim & Dataset
  - Results & Discussions
  - Threats to Validity
- Related Work
- Conclusion & Future Work
Outline

- **Background & Motivation**
- Properties of Local Variables
- Abnormality of Local Variables
- Empirical Study
  - Aim & Dataset
  - Results & Discussions
  - Threats to Validity
- Related Work
- Conclusion & Future Work
Local Variable

- **Local variables** are popular artifacts
- **All programmers from beginners to experts** usually use local variables in their code
  - The requirement specifications and the design documents usually **do not specify** the names of local variables
  - So, **programmers can freely decide** the names of local variables

**Giving a name** to a local variable is almost always at **the programmer’s discretion**
Diversity in Naming

For a local variable, different programmers may want to give it different names.

It causes a quality variation, especially in the readability and the comprehensibility.
More Descriptive Name is Better?

A longer and more descriptive name makes a local variable easier to understand its role.

Some studies or concerns ….

- There is no significant difference in comprehending the role of variable between fully-spelled name and abbreviated one\(^4\): e.g., index vs. idx.
- An experiment focusing on the human short-term memory showed that a longer name may degrade the readability of code\(^5\).
- Kernighan et al.\(^1\) said the it is overdone to give a long and descriptive name to a local variable with a narrow scope.
Research Motivation

- We have an interest in the effect of a local variable's name on the code quality.

What are the real trends of local variables' names (and scopes)?

Is an abnormal variable harmful to the quality of code?
Outline

- Background & Motivation
- **Properties of Local Variables**
- Abnormality of Local Variables
- Empirical Study
  - Aim & Dataset
  - Results & Discussions
  - Threats to Validity
- Related Work
- Conclusion & Future Work
Major Properties of Local Variable

- **Name**
- **Scope**
- **Type**

They are not independent each other; The decision of local variable's name is sometimes affected by its scope and type.
Scope vs. Name

When a variable's scope is narrow, a short and simple name may be acceptable; a long and descriptive name seems to be overdone\(^1\).

```
for(int i = 0; i < data.length; i++){
    System.out.println(i + "":" + data[i])
}
```

If the loop has a long and complicated block, it might be better to give a descriptive name.

```
for(int indexOfArray = 0; indexOfArray < data.length; indexOfArray+++
    System.out.println(indexOfArray + "":" + data[indexOfArray])
}
```
Type vs. Name

Some variables' names are from their types

Example: a part of common JDBC sample code

```
Connection con
    = DriverManager.getConnection("jdbc:....");

Statement stmt = con.createStatement();

String query = "select ....";

ResultSet rs = stmt.executeQuery(query);
```

A type (or class) name does not seem to be independent of the way of naming variable
Outline

• Background & Motivation
• Properties of Local Variables
• Abnormality of Local Variables
• Empirical Study
  ◦ Aim & Dataset
  ◦ Results & Discussions
  ◦ Threats to Validity
• Related Work
• Conclusion & Future Work
Preferred Features of Local Variable

Many coding standards (conventions) and practices say preferred features:

**Short name**

Local variable names can/should be short (Kernighan et al.[1], the Sun Java code convention[2], the GNU coding standard[3])

**Narrow scope**

Local variables should be declared at the points which minimize their scopes (The GNU coding standard[3], the Google Java style[7])
Research Questions (1/2)

First, we would like to understand the real trends of local variables in Java.

**RQ1**: What are the real trends of local variables in terms of their properties? (What are abnormal ones?)

Under RQ1, we can see the de facto standard for local variables, and what are abnormal ones.
Research Questions (2/2)

Then, we perform an analysis focusing on an impact of abnormal variable on a code quality.

RQ2: Is the presence of an abnormal local variable in a Java method related to the method’s change-proneness?

If the presence of an abnormal variable in a method is harmful to the quality of code, the method cannot survive unscathed after its release.
Procedure to Evaluate the Degree of Abnormality

1. **Classify** local variables into several categories **according to their types**

2. Express a variable as a **feature vector** \( \mathbf{x} \) whose elements correspond to the variable's properties

3. Compute the **Mahalanobis distance** between \( \mathbf{x} \) and their mean vector \( \mathbf{\mu} \): our measure of abnormality
1. Classification of Local Variables

- The **type of a variable** may have an affect on **its naming**: one of threats to validity
- We want to **avoid such an impact** on our results in order to mitigate this threat

<table>
<thead>
<tr>
<th>category</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) primitive type</td>
<td>primitive types such as int</td>
</tr>
<tr>
<td>(ii) primitive array</td>
<td>arrays of primitive-type variables</td>
</tr>
<tr>
<td>(iii) reference type</td>
<td>class or interface types  (except for ancestors of Exception, Error or Throwable)</td>
</tr>
<tr>
<td>(iv) reference array</td>
<td>arrays of class/interface-type variables</td>
</tr>
</tbody>
</table>
2. Feature Vector

For each local variable in each category, we consider a corresponding vector \( \mathbf{x} \):

\[
\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}
\]

- \( x_1 \): the number of characters comprising the variable's name
- \( x_2 \): the number of words comprising the variable's name
- \( x_3 \): the number of lines in which the variable is valid, i.e., scope
3. Measure of Abnormality (1/4)

- Let $\mu$ be the mean vector of $\{ x \}$ in a category.
- The distance between $x$ and $\mu$ seems to be a measure of $x$'s abnormality.

A simple measure (the Euclidean distance)

$$|x - \mu| = \sqrt{(x - \mu)^T (x - \mu)}$$

However, the Euclidean distance does not consider the dispersion of data and the correlations between their elements.
3. Measure of Abnormality (2/4): The Mahalanobis Distance

- The **Mahalanobis distance** considers the distribution of data.
- It is useful to **detect an abnormal object** in the given dataset.

The Mahalanobis distance

\[ d(x, \mu) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)} \]

\[ S = (\sigma_{ij}): \text{the variance-covariance matrix} \]
3. Measure of Abnormality (3/4): The Mahalanobis Distance

To get an intuitive understand, let us consider one-dimensional version

$$d(x, \mu) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)}$$

$$d(x, \mu) = \sqrt{(x - \mu) \frac{1}{\sigma^2} (x - \mu)} = \frac{|x-\mu|}{\sigma}$$

The Euclidian distance normalized by the standard deviation of data
3. Measure of Abnormality (4/4): The Mahalanobis Distance

We propose to evaluate an abnormality of an object by using its **Mahalanobis distance** from the center of dataset.

Their **Euclidian distances** are **the same**, but the **red one is clearly farther** from the center.

**Mahalanobis distance** can capture such a difference.
Outline

- Background & Motivation
- Properties of Local Variables
- Abnormality of Local Variables
- **Empirical Study**
  - Aim & Dataset
  - Results & Discussions
  - Threats to Validity
- Related Work
- Conclusion & Future Work
Empirical Study

- **Aim:**
  To analyze a relationship between an abnormal variable and the method quality to which the variable belongs (in terms of change-proneness)

- **Dataset:** six popular OSS products

<table>
<thead>
<tr>
<th>Product name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elasticsearch</td>
<td>developed in Java</td>
</tr>
<tr>
<td>Fastjson</td>
<td>managed with Git</td>
</tr>
<tr>
<td>Guava</td>
<td>popular (having many stars at GitHub)</td>
</tr>
<tr>
<td>libGDX</td>
<td>large-scale (&gt; 100 KLOC)</td>
</tr>
<tr>
<td>Presto</td>
<td></td>
</tr>
<tr>
<td>RxJava</td>
<td></td>
</tr>
</tbody>
</table>
Data Collection

1. Check the **change history of all source files**

2. Extract methods of each version of each source file: obtain the **change history of all methods**
1. Check the change history of all source files
2. Extract methods of each version of each source file: obtain the change history of all methods
3. Extract local variables from each method
4. Categorize local variables by their types, and compute their degrees of abnormalities by using the Mahalanobis distance from the center of data

\[ d(x, \mu) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)} \]

log transformation (if the distribution is skewed)
Outline

- Background & Motivation
- Properties of Local Variables
- Abnormality of Local Variables
- Empirical Study
  - Aim & Dataset
  - Results & Discussions
  - Threats to Validity
- Related Work
- Conclusion & Future Work
Research Question 1

RQ1: What are the real trends of local variables in terms of their properties? (What are abnormal ones?)

Under RQ1, we can see the de facto standard for local variables, and what are abnormal ones
Results: Popular Names

**Reference** type variables tend to have richer meaning names than **primitive** typed ones.

Names of **arrays** tend to be **plural forms** of words or just "array".

Surprisingly, **single-letter** names are **not so popular**.

<table>
<thead>
<tr>
<th>Reference type variables tend to have richer meaning names than primitive typed ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Names of arrays tend to be plural forms of words or just &quot;array&quot;</td>
</tr>
<tr>
<td>Surprisingly, single-letter names are not so popular</td>
</tr>
</tbody>
</table>

Categorizing variables by their types seems to be reasonable.
A majority of names consist of **less than 10 characters**

Names of **primitive types** seem to be **shorter** than the ones of other types, but their **differences are small** (1-2 characters)

### Name: rowGroupDictionaryLengthStreamCheckpoint

### Type: RowGroupDictionaryLengthStreamCheckpoint

### Scope: 1 line
### Results: Features of Variables (2/3)

#### Word Count of Name

<table>
<thead>
<tr>
<th>product</th>
<th>min</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elasticsearch</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Fastjson</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Guava</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>libGDX</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Presto</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>RxJava</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

#### Words (terms) are split according to the CamelCase or the SnakeCase

A majority of names are made from **single word (or term)**

The difference in type seems to have no impact on the feature

**Name**: canBeAllocatedToAtLeastOneNode  
**Type**: boolean  
**Scope**: 143 lines
### Results: Features of Variables (3/3)

**Scope (Valid Line Count)**

A majority of local variables are valid within blocks of **20 or less lines**

<table>
<thead>
<tr>
<th>product</th>
<th>min</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) primitive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>22</td>
<td>354</td>
</tr>
<tr>
<td>Fastjson</td>
<td>2</td>
<td>9</td>
<td>22</td>
<td>47</td>
<td>422</td>
</tr>
<tr>
<td>Guava</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>11</td>
<td>80</td>
</tr>
<tr>
<td>libGDX</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>26</td>
<td>479</td>
</tr>
<tr>
<td>Presto</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>15</td>
<td>142</td>
</tr>
<tr>
<td>RxJava</td>
<td>2</td>
<td>9</td>
<td>19</td>
<td>39</td>
<td>181</td>
</tr>
<tr>
<td>(ii) primitive array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>59</td>
</tr>
<tr>
<td>Fastjson</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>31</td>
<td>167</td>
</tr>
<tr>
<td>Guava</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>15</td>
<td>56</td>
</tr>
<tr>
<td>libGDX</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>21</td>
<td>478</td>
</tr>
<tr>
<td>Presto</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>17.5</td>
<td>74</td>
</tr>
<tr>
<td>RxJava</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii) reference</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>23</td>
<td>356</td>
</tr>
<tr>
<td>Fastjson</td>
<td>1</td>
<td>5</td>
<td>12</td>
<td>36</td>
<td>424</td>
</tr>
<tr>
<td>Guava</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>11</td>
<td>135</td>
</tr>
<tr>
<td>libGDX</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>23</td>
<td>475</td>
</tr>
<tr>
<td>Presto</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>16</td>
<td>161</td>
</tr>
<tr>
<td>RxJava</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>26</td>
<td>236</td>
</tr>
<tr>
<td>(iv) reference array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>2</td>
<td>6</td>
<td>14</td>
<td>25</td>
<td>85</td>
</tr>
<tr>
<td>Fastjson</td>
<td>2</td>
<td>8</td>
<td>20</td>
<td>34</td>
<td>405</td>
</tr>
<tr>
<td>Guava</td>
<td>2</td>
<td>4.5</td>
<td>8</td>
<td>14</td>
<td>114</td>
</tr>
<tr>
<td>libGDX</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>18</td>
<td>212</td>
</tr>
<tr>
<td>Presto</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>17.5</td>
<td>76</td>
</tr>
<tr>
<td>RxJava</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>21</td>
<td>239</td>
</tr>
</tbody>
</table>

The difference in type seems to have no impact on the feature

**max sample**

Name: lenMain  
Type: int  
Scope: 479 lines
Answer to RQ1

RQ1: What are the **real trends** of local variables in terms of their properties? (What are **abnormal** ones?)

- Non-compound **short names** (a few or less characters) are major
- A variable's **scope is about 20 or less lines** regardless of variable's type

Many variables seem to **conform to the common coding conventions** in terms of the **length of name**; **Scopes** might be considered **less important**
Research Question 2

RQ2: Is the presence of an abnormal local variable in a Java method related to the method’s change-proneness?

If the presence of an abnormal variable in a method is harmful to the quality of code, the method cannot survive unscathed after its release.
A Java method may have **two or more local variables**

We have to **link their evaluations (abnormalities) to their method**

i. The **summation** of abnormalities

ii. The **maximum** of abnormalities
**Concern about Confounding Factor**

We had a concern that the method size (LOC) may be a confounding factor.

- Spearman's correlation coefficient

<table>
<thead>
<tr>
<th>project</th>
<th>(i) sum</th>
<th>(ii) max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elasticsearch</td>
<td>0.555</td>
<td>0.206</td>
</tr>
<tr>
<td>Fastjson</td>
<td>0.721</td>
<td>0.331</td>
</tr>
<tr>
<td>Guava</td>
<td>0.499</td>
<td>0.224</td>
</tr>
<tr>
<td>libGDX</td>
<td>0.581</td>
<td>0.082</td>
</tr>
<tr>
<td>Presto</td>
<td>0.438</td>
<td>0.137</td>
</tr>
<tr>
<td>RxJava</td>
<td>0.471</td>
<td>0.038</td>
</tr>
</tbody>
</table>

The sum of abnormalities seems to be positively correlated to the size of method (LOC).

We decided to use "(ii) max" in our analysis.
Category of Method in Terms of the Variable's Abnormality

We classified methods into three categories in accordance with the level of abnormality:

- **Low level**: $M_0$
  - $\text{ma}(m) < 1$

- **Middle level**: $M_1$
  - $1 \leq \text{ma}(m) < 2$

- **High level**: $M_2$
  - $\text{ma}(m) \geq 2$

Cf. $|x - \mu| < \sigma$

$\sigma \leq |x - \mu| < 2\sigma$

$|x - \mu| \geq 2\sigma$

\[\text{ma}(m) : \text{maximum abnormality in method } m\]
Results: Scatter Diagram

\( M_0 \) \( M_1 \) \( M_2 \)

Top 25% of 
\#(code changes)

Result of Fastjson
Rate of Change-Prone Methods

We compute the rates of change-prone methods in each set $M_i$ (for $i = 0, 1, 2$) as

$$\frac{H_i}{L_i + H_i}$$
Results: Rates of Change-Prone Methods

$M_2$ (high level abnormality) showed the highest rates for all products.
Results: $M_0$ vs. $M_2$

$M_2$ (high level abnormality) has about 1.2 – 2.5 times higher rate than $M_0$ (low level abnormality)

libGDX: $p = 0.0456; 1 - \beta = 0.552$

others: $p < 0.01; 1 - \beta > 0.8$
RQ2: Is the presence of an abnormal local variable in a Java method related to the method’s change-proneness?

- A method having a highly-abnormal local variable is 1.2 – 2.5 times more likely to be change-prone than the one having only normal variables

The presence of an abnormal local variable seems to be related to the method's change-proneness; however, we need a further analysis to see the reasons and the actual impacts
Outline

• Background & Motivation
• Properties of Local Variables
• Abnormality of Local Variables
• Empirical Study
  ◦ Aim & Dataset
  ◦ Results & Discussions
  ◦ Threats to Validity
• Related Work
• Conclusion & Future Work
Threats to Validity (1/2)

Our selection of subjects:

- We analyzed large-scale Java OSS products
- Products written in another language, small-sized products or commercial products may show different trends

The notion of local variable is common to many modern programming language, so the difference in language may not be a serious threat.

Since our dataset is organized from some different domains, the threat to the generality is mitigated.
A further analysis with more products is our future work.
Threats to Validity (2/2)

- Details of code changes:
  - We have just checked the number of code change events
  - We did not investigate the impacts of changes

- Threshold of the Mahalanobis distance
  - We simply used the notion of 1-sigma and 2-sigma in the normal distribution
  - We need a detailed analysis on the threshold to effectively decide abnormal variables

Further analyses from these perspectives are our significant future work
Outline

• Background & Motivation
• Properties of Local Variables
• Abnormality of Local Variables
• Empirical Study
  ◦ Aim & Dataset
  ◦ Results & Discussions
  ◦ Threats to Validity
• Related Work
• Conclusion & Future Work
Related Work

- The following studies reported that an identifier with a longer name is related to poor quality
  - Binkley et al. [5] studied a relationship between the human short-term memory and the length of identifier through an empirical study with 158 programmers
  - Kawamoto et al. [12] analyzed identifiers used in Eclipse and NetBeans

Their results are supporting our results; we focused on not only names but also their types and scopes
Outline

- Background & Motivation
- Properties of Local Variables
- Abnormality of Local Variables
- Empirical Study
  - Aim & Dataset
  - Results & Discussions
  - Threats to Validity
- Related Work
- Conclusion & Future Work
Conclusion

We proposed to evaluate an abnormality of a local variable by using the Mahalanobis distance, conducted an empirical analysis, and reported:

- It is worthwhile to classify local variables by their types for an accurate analysis
- The majority of local variables have short names and narrow scopes; a name is often a single word or its abbreviation
- Java methods having abnormal local variables do not tend to survive unscathed after their releases
Future Work

- Since we have just reported trends, we need further analyses of the impacts of abnormal local variables on the code quality.

- Further analyses focusing on the followings are important future work:
  - details of code changes
  - changes of abnormalities through upgrades
  - meaning of words in variables’ names
  - the author information
  - the conventions adopted by the development team