A Longitudinal Study of Anti Micro Patterns in 113 versions of Tomcat

G. Destefanis, S. Qaderi, D. Bowes, J. Petrić
University of Hertfordshire
Hatfield, UK
{g.destefanis,d.h.bowes,j.petric}@herts.ac.uk

M. Ortu
DIEE, University of Cagliari
Cagliari, Italy
marco.ortu@diee.unica.it

ABSTRACT

Background: Micro patterns represent design decisions in code. They are similar to design patterns and can be detected automatically. These micro structures can be helpful in identifying portions of code which should be improved (anti-micro patterns), or other well-designed parts which need to be preserved. The concepts expressed in these design decisions are defined at class-level; therefore the primary goal is to detect and provide information related to a specific granularity level. Aim: this paper aims to present preliminary results about a longitudinal study performed on anti-micro pattern distributions over 113 versions of Tomcat. Method: we first extracted the micro patterns from the 113 versions of Tomcat, then found the percentage of classes matching each of the six anti-micro pattern considered for this analysis, and studied correlations among the obtained time series after testing for stationarity, randomness and seasonality. Results: results show that the time series are stationary, not random (except for Function Pointer), and that additional studies are needed for studying seasonality. Regarding correlations, only the Pool and Record time series presented a correlation of 0.69, while moderate correlation has been found between Function Pointer and Function Object (0.58) and between Cobol Like and Pool (0.44).

KEYWORDS

micro patterns, time series analysis, software engineering

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION

Gil and Maman [15] introduced the concept of micro patterns as design decisions in Java, somehow similar to design patterns, but at a lower level of abstraction. Design patterns are difficult to detect from source code using automated tools and represent a general concept or methodology used for designing a piece of software. Micro patterns are defined at class level and can capture good and bad programming practices, spanning from data encapsulation and inheritance to coding practices typical of procedural programming. Singer et al. [22] presented a catalog of 17 nano-patterns which are categorized into 4 groups. Codalab et al. [6], extracted micro patterns and nano-patterns from three versions of Tomcat and other two systems and compared their distributions in code smell versus non-code smell classes and methods, concluding that code smells are correlated with both micro and nano-patterns. Micro patterns have been proven useful also for detecting software vulnerabilities [23, 24]. In this work, we present a preliminary study of the evolution (in terms of quantity) of six micro patterns of the catalog introduced by Gil and Maman [13] for 113 versions of Tomcat 1, an open source Java Servlet Container developed by the Apache Software Foundation (from version 3.3.2 to version 8.0.9) and heavily used in software engineering research [20]. We enriched the dataset provided by Destefanis et al. [8], detecting the micro patterns from the source code of each version of Tomcat. The new dataset is openly available at the following link https://bitbucket.org/giuseppedestefanis/promise2018 and contains 113 new tables with all the micro patterns for each version of Tomcat. We considered six micro patterns which are related to bad programming practices and which have already been analyzed in previous studies [1, 7, 10, 11, 17]. Here we define each micro pattern considered in this study (definitions provided by Gil at al. [13]) and the motivations which explain why they are considered anti-patterns.

Cobol-like: a class with a single static method, but no instance member. As explained by Arcelli et al. [1], a class matching this pattern is not designed following the object-oriented paradigm, and can be frequent in code developed by beginners.

Function Pointer: a class with a single public instance method, but with no fields. Classes matching this pattern are not necessarily badly designed, but they considered the object-oriented equivalent of function pointers in procedural programming.

Record: a class in which all fields are public, no declared methods. Classes matching this pattern do not respect encapsulation, a principle according to which fields should be declared private and managed with getter and setter methods.

Function Object: a class with a single public instance method, and at least one instance field. Classes matching this pattern fall in the micro pattern category called “degenerate behavior”. It is similar

1http://tomcat.apache.org
to the Function Pointer micro pattern, but Function Object has instance fields instead. An instance of a class which matches the definition of Function Object can store parameters to the main method of the class. **Pseudo Class**: a class which can be rewritten as an interface: no concrete methods, only static fields. Classes matching this pattern should be refactored and rewritten as interfaces.

We built six time series, one for each anti-pattern, which represent the percentage of occurrence of a given pattern in every release of Tomcat. The obtained time series have 113 points. In this preliminary work, we only considered production classes (e.g., we excluded all the test classes from each release). Time series are heavily used for predictive studies and provide information about trends and seasonality. We first studied the six time series for stationarity and calculated the cross-correlation coefficient among all the anti-patterns.

### 2 METHODOLOGY AND RESULTS

In this study, we were interested in analysing randomness, seasonality and evaluating the cross-correlation of six antipatterns of the catalog proposed by Gil and Maman [13].

To analyze such properties, we considered an observation time of one release. Since in the Tomcat dataset [8], there are 113 releases, we obtained 113 points for the six time series, and measured the percentage of production classes matching the six anti-patterns.

The six time series are presented in Fig. 2, with a linear trend line (in red) which provides a visual indication of the presence of the patterns over time. Software developers should aim at reducing the percentage of these anti-patterns. If the presence of an anti-pattern grows over time, managers and developers should take action to reverse the condition. Visualizing the six anti-pattern time series (Fig. 2) provides useful information about the evolution of the system. Additionally, it is possible to see that the percentage of the classes matching the six anti-patterns is low. Only Pool, Function Pointer and Function Object reached values above 8%, but only for one release. Cobol-Like and Pseudo Class reached a maximum of 1.2%, while Record reached a maximum of only 1%. Before studying the cross-correlation among the six anti-patterns and analyzing randomness and seasonality, we studied the time series for stationarity. A time series is stationary if autocorrelation, variance, expectation, do not vary with time [14, 18, 19]. Stationarity is a condition required for being able to perform predictive analysis on a time series.

We used the R environment and we applied the Ljung-Box test [16]: this test for stationarity confirms the independence of increments, where rejection of the null hypothesis $H_0$ suggests stationarity (the null hypothesis $H_0$ is that the data are non-stationary). The results of the test is shown in Table 1. The cells in green indicate that the p-value for the corresponding test is below 5% (our cutoff for significance); thus we infer in these cases that the test indicates stationarity.

If the time-series under study are stationary, it is possible to calculate the cross correlation. The cross correlation function (ccf) is defined as the set of correlations (height of the vertical line segments in Fig. 1) between two time series $x_t$ and $y_t$ for lags $h = 0, 1, 2, \ldots$. A negative value for $h$ represents a correlation between the $x$-series at a time before $t$ and the $y$-series at time $t$

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Ljung-Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cobol Like</td>
<td>4.41e-16</td>
</tr>
<tr>
<td>Pool</td>
<td>3.15e-05</td>
</tr>
<tr>
<td>Function Pointer</td>
<td>9.329e-06</td>
</tr>
<tr>
<td>Record</td>
<td>3.256e-09</td>
</tr>
<tr>
<td>Function Object</td>
<td>6.883e-15</td>
</tr>
<tr>
<td>Pseudo Class</td>
<td>2.2e-16</td>
</tr>
</tbody>
</table>

Table 1: Stationarity test results

For example, if the lag $h = -1$, then the cross correlation value would give the correlation between $x_{t - 1}$ and $y_t$. On the contrary, negative lines correspond to anti-correlated events.

The ccf helps to identify lags of $x_t$ that could be predictors of the $y_t$ series.

- When $h < 0$, $x$ leads $y$.
- When $h > 0$, $y$ leads $x$.

![Figure 1: Pool and Cobol-like Cross-Correlation](https://cran.r-project.org/web/packages/randtests/randtests.pdf)

As shown in Table 2, the highest correlation of 0.69 (at lag 0) is between the patterns Pool and Record. A correlation of 0.58 (at lag -10) exists between the patterns Function Pointer and Function Object, while a correlation of 0.44 (at lag 17) exists between the patterns Cobol-like and Pool. The highest correlation (Pool-Record), at lag 0, confirm the similarity between the two patterns (which is possible to appreciate from the two definitions), but highlights the fact that even if the correlation value is not negligible, the two time series brings different information. A time series is considered random if it consists of independent values from the same distribution. We used the Bartels test [3] for studying randomness [5]. We used the R package `randtest` and applied Bartels test, in which the null hypothesis $H_0$ of randomness is tested against non-randomness. For studying the seasonality of the time series, we considered the **Augmented Dickey Fuller test** [2, 12, 21] in which the null hypothesis $H_0$ is that the data are non-seasonal, and the **Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test** [15]: the null
A Longitudinal Study of Anti Micro Patterns in 113 versions of Tomcat

PROMISE’18, October 10, 2018, Oulu, Finland

Figure 2: Time series
hypothesis $H_0$ is that the data are non-seasonal. The results of the tests are shown in Tables 3. For randomness, the cell in red indicates that the p-value for the corresponding test is higher than our cutoff for significance (5%); thus we infer in these cases that the test indicates randomness (null hypothesis $H_0$ of randomness). On the contrary, cells in white indicate that the p-value for the corresponding test is lower than 5% (our cutoff for significance); thus we infer in these cases that the test rejects the null hypothesis $H_0$ of non-seasonality; hence the time series are considered seasonal.

**Table 3: Randomness and seasonality test results**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Randomness Bartels-rank p-value</th>
<th>Seasonality Ang. D-F p-value</th>
<th>KPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cobol Like</td>
<td>2.2e-16</td>
<td>0.001091</td>
<td>0.01</td>
</tr>
<tr>
<td>Pool</td>
<td>0.3404</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Function Pointer</td>
<td>5.602e-15</td>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>Record</td>
<td>8.724e-07</td>
<td>0.014136</td>
<td>0.01</td>
</tr>
<tr>
<td>Pseudo Class</td>
<td>2.2e-16</td>
<td>0.0822</td>
<td>0.01</td>
</tr>
</tbody>
</table>

3 CONCLUSIONS

In this preliminary work, we studied the time series of six anti-micro patterns for 113 releases of Tomcat. Results show that the time series are stationary, not random (except for Function Pointer) and that further investigations are needed for seasonality. Regarding correlations, only the Pool and Record time series presented a correlation of 0.69, while moderate correlation has been found between Function Pointer and Function Object (0.58) and between Cobol Like and Pool (0.44). Time series are useful for predictive analysis, and for future works, we plan to study the time series of every micro pattern of the catalog and to study if micro patterns can be used for defect prediction purposes. The main limitation of this study is that we only considered Tomcat as the subject for our investigation, and this might affect the generalization of our results. Additional systems must be considered for future studies.

REFERENCES