# 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 studied 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

**ACM Reference Format:**
G. Destefanis, S. Qaderi, D. Bowes, J. Petrić and M. Ortu. 2018. A Longitudinal Study of Anti Micro Patterns in 113 versions of Tomcat. In *The 14th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'18), October 10, 2018, Oulu, Finland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3273934.3273945

## 1 INTRODUCTION

Gil and Maman [13] 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.


*PROMISE'18, October 10, 2018, Oulu, Finland*

ACM ISBN 978-1-4503-6593-2/18/10…$15.00
https://doi.org/10.1145/3273934.3273945

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. Codabux 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.

**Pool:** *a class which declares only static final fields, but no methods.* This pattern is also known as *constant interface (anti) pattern* [4], and it describes the practice of implementing interfaces which contain definitions of constants.

**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

[1] http://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, on 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 $x_t + h$ and $y_t$ for lags $h = 0, \pm1, \pm2, \ldots$. 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 1: Stationarity test results**

| Pattern | Ljung-Box |
|---|---|
| Cobol Like | 4.441e-16 |
| Pool | 5.155e-05 |
| Function Pointer | 9.329e-06 |
| Record | 3.256e-09 |
| Function Object | 6.883e-15 |
| Pseudo Class | 2.2e-16 |

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 $x$.

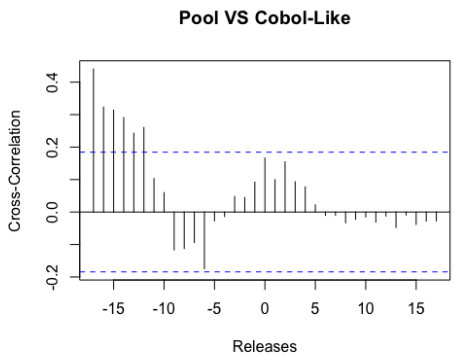
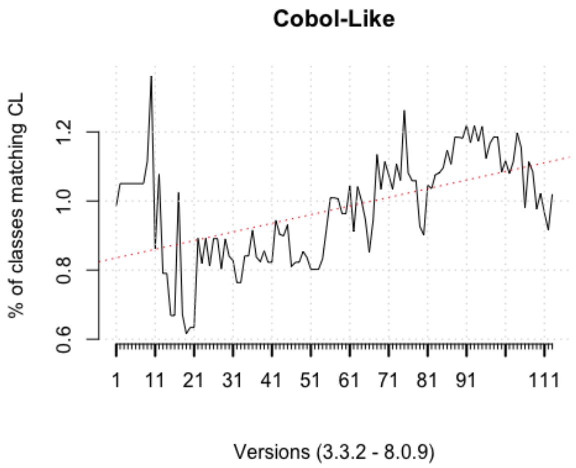


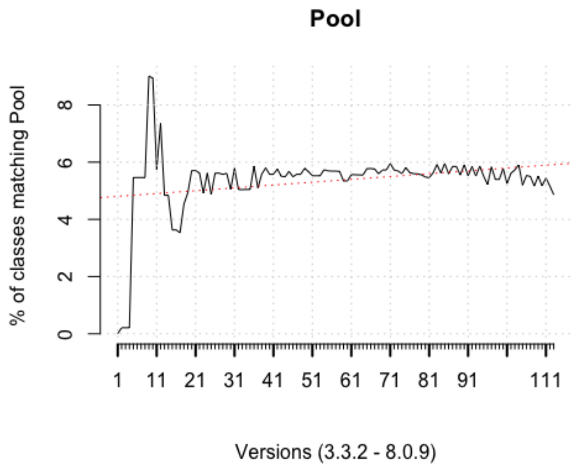

**Figure 1: Pool and Cobol-like Cross-Correlation**

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*[2] 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

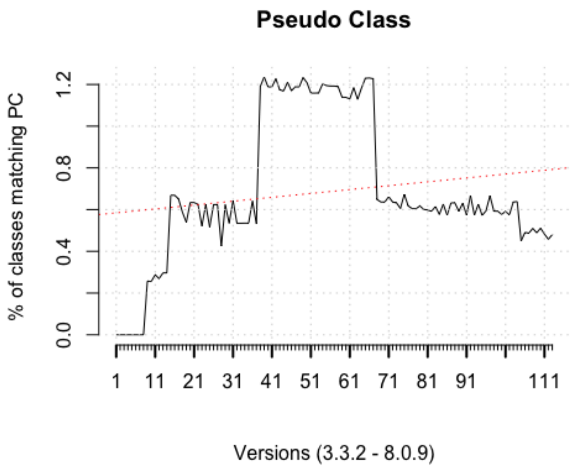
[2]https://cran.r-project.org/web/packages/randtests/randtests.pdf



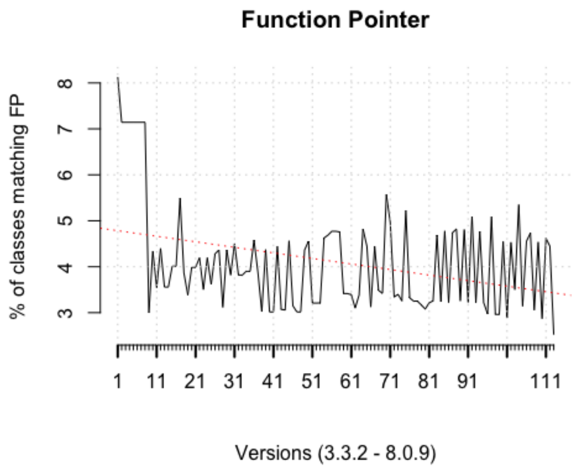
(a) Presence of Cobol Like over the 113 Tomcat releases



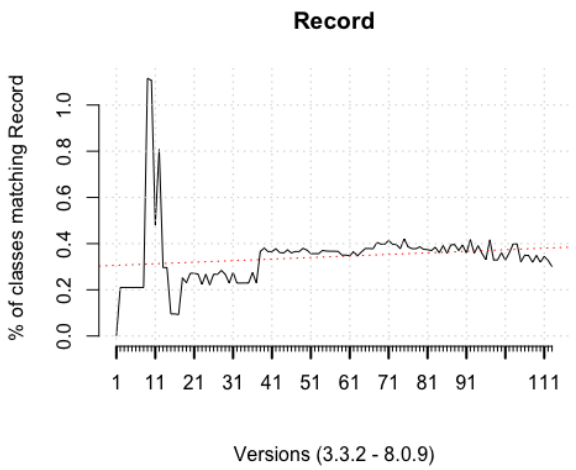
(b) Presence of Function Pointer over the 113 Tomcat releases



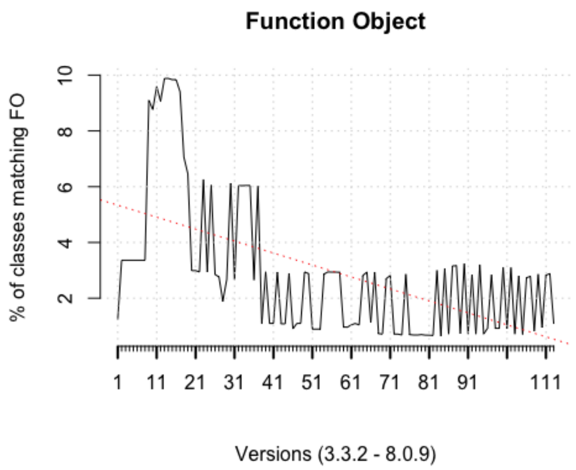
(c) Presence of Pseudo Class over the 113 Tomcat releases



(d) Presence of Function Pointer over the 113 Tomcat releases



(e) Presence of Record over the 113 Tomcat releases



(f) Presence of Function Object over the 113 Tomcat releases

Figure 2: Time series

**Table 2: Cross correlation among time series**

| | CL | P | FP | R | FO | PC |
|---|---|---|---|---|---|---|
| CL | - | 0.44 - {17} | 0.18 - {0} | 0.41 - {0} | 0.028 - {-13} | 0.38 - {17} |
| P | - | - | 0.18 - {4} | 0.69 - {0} | 0.1 - {-4} | 0.31 - {-3} |
| FP | - | - | - | 0.29 - {-8} | 0.58 {-10} | 0.005 - {8} |
| R | - | - | - | - | 0.15 - {-5} | 0.28 - {12} |
| FO | - | - | - | - | - | 0.02 - {-17} |
| PC | - | - | - | - | - | - |

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%. For seasonality, the cells in green indicate that the p-value for the corresponding test is less 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**

| | *Randomness* | *Seasonality* | |
|---|---|---|---|
| **Pattern** | **Bartels-rank p-value** | **Aug. D-F p-value** | **KPSS** |
| Cobol Like | 2.2e-16 | 0.3151 | 0.01 |
| Pool | 0.001091 | 0.01 | 0.1 |
| Function Pointer | 0.4604 | 0.01 | 0.01 |
| Record | 5.602e-15 | 0.01 | 0.1 |
| Function Object | 8.724e-07 | 0.1641 | 0.01 |
| Pseudo Class | 2.2e-16 | 0.5822 | 0.01 |

**Table 3** shows the results for all the time series of each anti-micro pattern. For five time series, the hypothesis of randomness is rejected. Only for Function Pointer, the Bartels test indicates randomness. For the Augmented Dickey-Fuller test, Pool, Function Pointer and Record have a time series which presents seasonality, while Cobol Like, Function Object and Pseudo Class are not seasonal. For the KPSS test, the null hypothesis of non-seasonality is rejected for Pool and Record. We considered two tests for seasonality, following the same procedure adopted by Destefanis et al. [9]. Since there is discordance between the two tests in five cases over six, it would be necessary to perform additional studies, using, for example, Fast Fourier Transform (FFT).

## 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


[1] Francesca Arcelli and Stefano Maggioni. 2009. Metrics-based detection of micro patterns to improve the assessment of software quality. In *Proceedings of the 1st Symposium on Emerging Trends in Software Metrics*. 50–59.

[2] Anindya Banerjee, Juan J Dolado, John W Galbraith, David Hendry, and others. 1993. Co-integration, error correction, and the econometric analysis of non-stationary data. *OUP Catalogue* (1993).

[3] Robert Bartels. 1982. The rank version of von Neumann's ratio test for randomness. *J. Amer. Statist. Assoc.* 77, 377 (1982), 40–46.

[4] Joshua Bloch. 2001. Effective Java: Programming Language Guide. Java series. (2001).

[5] Peter J Brockwell and Richard A Davis. 2006. *Introduction to time series and forecasting*. Springer Science & Business Media.

[6] Zadia Codabux, Kazi Zakia Sultana, and Byron J Williams. 2017. The relationship between traceable code patterns and code smells. In *Proc. 29th Int. Conf. Software Engineering and Knowledge Engineering*.

[7] Giulio Concas, Giuseppe Destefanis, Michele Marchesi, Marco Ortu, and Roberto Tonelli. 2013. Micro patterns in agile software. In *International Conference on Agile Software Development*. Springer, 210–222.

[8] Giuseppe Destefanis, Mahir Arzoky, Steve Counsell, Stephen Swift, Marco Ortu, Roberto Tonelli, and Michele Marchesi. 2018. 113 times Tomcat: A dataset. *PeerJ Preprints* 6 (2018), e26491v1.

[9] Giuseppe Destefanis, Marco Ortu, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. 2017. On the randomness and seasonality of affective metrics for software development. In *Proceedings of the Symposium on Applied Computing*. ACM, 1266–1271.

[10] Giuseppe Destefanis, Roberto Tonelli, Giulio Concas, and Michele Marchesi. 2012. An analysis of anti-micro-patterns effects on fault-proneness in large java systems. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 1251–1253.

[11] Giuseppe Destefanis, Roberto Tonelli, Ewan Tempero, Giulio Concas, and Michele Marchesi. 2012. Micro pattern fault-proneness. In *Software engineering and advanced applications (SEAA), 2012 38th EUROMICRO conference on*. IEEE, 302–306.

[12] Francis X Diebold and Glenn D Rudebusch. 1991. On the power of Dickey-Fuller tests against fractional alternatives. *Economics letters* 35, 2 (1991), 155–160.

[13] Joseph Yossi Gil and Itay Maman. 2005. Micro patterns in Java code. *ACM SIGPLAN Notices* 40, 10 (2005), 97–116.

[14] James Douglas Hamilton. 1994. *Time series analysis*. Vol. 2. Princeton university press Princeton.

[15] Denis Kwiatkowski, Peter CB Phillips, Peter Schmidt, and Yongcheol Shin. 1992. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of econometrics* 54, 1 (1992), 159–178.

[16] Greta M Ljung and George EP Box. 1978. On a measure of lack of fit in time series models. *Biometrika* 65, 2 (1978), 297–303.

[17] Marco Ortu, Giuseppe Destefanis, Matteo Orru, Roberto Tonelli, and Michele L Marchesi. 2015. Could micro patterns be used as software stability indicator?. In *Patterns Promotion and Anti-patterns Prevention (PPAP), 2015 IEEE 2nd Workshop on*. IEEE, 11–12.

[18] Maurice Bertram Priestley. 1981. Spectral analysis and time series. (1981).

[19] Maurice Bertram Priestley. 1988. Non-linear and non-stationary time series analysis. (1988).

[20] Brian Robinson and Patrick Francis. 2010. Improving industrial adoption of software engineering research: a comparison of open and closed source software. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 21.

[21] Said E Said and David A Dickey. 1984. Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika* 71, 3 (1984), 599–607.

[22] Jeremy Singer, Gavin Brown, Mikel Luján, Adam Pocock, and Paraskevas Yiapanis. 2010. Fundamental nano-patterns to characterize and classify java methods. *Electronic Notes in Theoretical Computer Science* 253, 7 (2010), 191–204.

[23] Kazi Zakia Sultana. 2017. Towards a software vulnerability prediction model using traceable code patterns and software metrics. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 1022–1025.

[24] Kazi Zakia Sultana and Byron J Williams. 2017. Evaluating micro patterns and software metrics in vulnerability prediction. In *Software Mining (SoftwareMining), 2017 6th International Workshop on*. IEEE, 40–47.