Code Smells and Micro Patterns Dependencies

Francesca Arcelli Fontana[1], Marco Zanoni[1], and Bartosz Walter[2]

[1] University of Milano-Bicocca, 20126 Milano, Italy,
\{arcelli,marco.zanoni\}@disco.unimib.it,
WWW home page: http://essere.disco.unimib.it
[2] Pozna University of Technology, Faculty of Computing, Pozna, Poland,
bartosz.walter@cs.put.poznan.pl

Abstract. To better support software maintenance and evolution, it is important to evaluate the quality of a system, identifying defects, code smells and anti patterns as hints pointing to subsystems that require improvement. In this paper we analyze the possible dependencies existing among code smells and between code smells and micro patterns – structural characteristics capturing common programming techniques. Knowledge of possible relations could improve the detection techniques of code smells and provide hints on how to improve the quality of a system. We discover these relations in an experiment employing machine learning techniques.

Key words: source code quality evaluation, micro patterns, code smell detection

1 Introduction

Many works have been published on the role of metrics in design quality evaluation, concerning different aspects and perspectives [1, 2, 3]. Some of these metrics, like those described in the work of Lanza and Marinescu [4], are also commonly used for code smells detection.

Code smells are characteristics of the software that may indicate an implementation or design problem which further makes code maintenance more difficult. An initial set of smells has been defined by Fowler [5], other have been identified later and new ones can be discovered.

Several factors coming from various data sources are useful as inputs for detection algorithms. Most of them are based on the composition of selected of metrics, either standard object-oriented metrics or defined ad hoc. Use of metrics raises the issue of calibrating the detectors by defining appropriate thresholds. It is important to notice that code smells are by nature subjective: their reception, evaluation and interpretation differ, depending on the sensitivity of the detection tool. This observation significantly affects the results of the detection process.

Metrics are the primary source of information about code quality. However, there are code smells that cannot be correctly captured only with metrics and require more data for making an accurate diagnosis. For example, specific code properties can be extracted only by analysis of the code structure, which is not subject to measurement, but rather factual constatation. Moreover, some symptoms are misleading when interpreted in isolation and should be analyzed with additional data, e.g. knowledge of other code smells that have already been identified.

The structural information about the code can be represented by micro patterns, proposed as a method of capturing some very common programming techniques. They can be thought of as class-level traceable patterns that can be automatically and unambiguously recognized. The existence of micro patterns can provide hints on good or bad programming practices. Some micro patterns directly correspond to critical classes that need to be refactored or to some object-oriented anti patterns.

Several empirical studies that correlate code smells with other software properties have been presented in the literature [8, 9]. However, to the best of our knowledge, no study had previously considered relations of code smells and micro patterns.

The aim of this paper is to identify and examine, on experimental basis, dependencies between micro patterns and code smells and among various code smells. We analyze if the presence of specific micro patterns is correlated with the presence of a particular code smell and if micro patterns can be used for more accurate code smells detection; we also discover relations existing between different code smells.

With this objectives, we:

- analyze the results obtained with the detection of micro patterns and code smells on different systems;
- manually analyze the relations discovered between them;
- discover statistical dependencies among certain code smells and micro patterns, using data mining algorithms.

We focus our analysis on software systems written in Java due to its preavalence and popularity.

The contributions of this work are organized as follows:

- identification of code smells and micro patterns using two tools (inFusion and PMD) on five versions of GanttProject and one of JHotDraw systems; (Section 2);
- analysis of the detection results aimed to discover interesting relations between micro patterns and code smells through manual inspection and the use of machine learning techniques (Section 5);
- analysis of the detection results, in attempt to discover interesting correlations between different code smells (Section 4).

2 Code Smells and Micro Patterns Detection

Code smells were introduced by Fowler and Beck [5] as generic, high level characteristics that can point to common flaws at the implementation and design level.

Unlike plain software metrics, they reflect intuitive perception of code quality, which makes them attractive for developers. However, their ambiguity and internal subjectivity make them very difficult to express them in a formal way and define appropriate and accurate detection algorithms for them.

For example, the Large Class code smell is featured by a class that "does too much" [5]. This description is intentionally vague and makes no references to the size, complexity or any other metric. Interpretation of it belongs to individual programmers, who make the evaluation based on their intuition and experience. Therefore, it is difficult to transform this notion into a detection rule.

Several tools for detecting various code smells have been developed (e.g. CodeWizard [10], JDeodorant [11], iPlasma [12, 4], CheckStyle [13], Stench Blossom [14]). In our research we decided to use InFusion and PMD, based on our previous experience.

These tools adopt slightly different strategies for detecting code smells. InFusion uses both rules and metrics, as described in the book of Lanza and Marinescu [4], and is capable of detecting several code smells. PMD, in contrast, detects only few smells, based on metrics aggregation only. PMD also allows for user-defined thresholds for metrics used in detection rules.

In Table 2 we report the detected code smells with their respective acronyms and the detection tool we used for them.

As the codebase for analysis we chose two open-source projects: GanttProject (5 versions) and JHotDraw (1 version). They are systems of medium size, composed of 393-549 classes for GanttProject and 800 classes for JHotDraw. We collected data both automatically and manually. First, we detected micro patterns and code smells using respective detectors. Next, we manually verified co-existence of both structures in the code. The obtained data was then processed separately for discovering code smell dependencies and code smell-micro pattern relations.

In Table 3 we briefly report the numbers of smell-crippled classes in the analyzed systems. The detailed description of the smells and their typical symptoms are given by Fowler [5].

Micro patterns [6] were proposed in order to capture very common, recurring programming techniques based on the static analysis of the program's code. Currently, 27 micro patterns are identified, subdivided into eight categories. The original paper contains a complete description and discussion on their relevance. The identified micro patterns instances can help to identify classes of particular interest within the system, e.g. classes similar to class-level antipatterns. For example, the *Data manager* micro pattern represents a class with only setters or getters. Detection of such instances allows for the identification of classes whose sole objective is to provide a repository for data along with basic manipulation routines.

To best of our knowledge, currently there are two tools for detecting micro patterns: a prototype software developed by Gil and Maman based on the byte code analysis [6], and the Micro Structures Detector of MARPLE (Metrics and Architecture Reconstruction PLugin for Eclipse) [15, 16], based on source code

Table 1. GanttProject and JHotDraw: size of releases

	GanttProject version					JHD
	1.10	1.10.1	1.10.2	1.10.3	1.11.1	7.0.9
Packages	27	27	27	28	33	94
Classes	393	395	396	402	549	800
Methods	1913	1921	1924	1937	2724	5573

Table 2. Code smells legend

	Name	Acronym	Detected by
GanttProject & JHotDraw	Brain Method	BM	InFusion
	Intensive Coupling	IC	InFusion
	Dispersed Coupling	DC	InFusion
	Significant Duplication	SD	InFusion
	God Class	GC	InFusion
	Data Class	DaC	InFusion
	Feature Envy	FE	InFusion
	Speculative Generality	SG	InFusion
	Long Method	LM	PMD
	Large Class	LC	PMD
	Long Parameter List	LPL	PMD
JHotDraw	Brain Class	BC	InFusion
	Refused Bequest	RB	InFusion
	Missing Template Method	MTM	InFusion
	Shotgun Surgery	SS	InFusion

analysis. In our analysis we used the latter one. In Table 4 we report the micro patterns instances detected in the analyzed systems.

3 Mining association rules

Association rules mining is a method of discovering relations between variables in large data sets. They are best suited to find statistically-supported dependencies within sets of objects (called itemsets). The generated rules indicate that a subset of condition attributes is likely to be related with decision attributes, and are described by two main parameters. *Support* is defined as the fraction of transactions in entire data set which contain the considered itemset X. *Confidence* is the estimation of the accuracy of the rule [17], or the percentage of times the rule will correctly describe new data, when it will be applied to new cases; it can be interpreted as a measure of trust in the discovered rule.

In this work we decided to employ *Apriori* and *Predictive APriori* [17] algorithm, implemented in the Weka tool.

Table 3. Code smells in GanttProject and JHotDraw detected by InFusion and PMD

	GanttProject					JHD
Smell	1.10.0	1.10.1	1.10.2	1.10.3	1.11.1	7.0.9
Brain Method	35	35	35	34	24	63
Intensive Coupling	27	27	27	28	29	41
Dispersed Coupling	0	0	0	5	1	0
Significant Duplication	48	46	44	42	18	392
Brain Class	0	0	0	0	0	8
God Class	11	11	11	11	13	14
Data Class	38	39	39	39	50	11
Feature Envy	7	8	8	8	8	24
Refused Bequest	0	0	0	0	0	7
Shotgun Surgery	0	0	0	0	0	10
Missing Template Method	0	0	0	0	0	21
Speculative Generality	8	9	9	9	8	0
Long Method	47	47	45	46	55	132
Large Class	9	10	9	9	12	32
Long Parameter List	67	66	65	65	81	186

Table 4. Micro patterns detected in the analyzed systems

	GanttProject					JHD
Micro pattern	1.10.0	1.10.1	1.10.2	1.10.3	1.11.1	7.0.9
Overrider	77	76	76	78	123	111
Sink	207	196	209	213	315	386
Function pointer	31	32	32	34	42	34
Function object	37	38	38	38	49	57
Immutable	68	68	24	24	33	104
Outline	5	5	68	70	36	12
Data manager	21	20	5	5	108	7
Compound box	17	17	22	22	10	21
State machine	13	13	17	17	24	14
Record	5	5	14	15	19	14
Extender	22	21	5	5	7	19
Trait	1	1	1	1	2	2

4 Code Smells associations

Distinct code smells correspond to different anomalies. However, some of them exhibit certain similarities, share common features or source flaws. Code smells can indicate other smells or – on the contrary – exclude them. Fowler observed an obvious relation existing between Duplicated Code and Large Class smells: “When a class has too many instance variables, duplicated code cannot be far behind”. Similarily, an object identified as a Large Class instance cannot be simultaneously a Lazy Class. Other dependencies are easy to notice based on the

Table 5. Best CS→CS rules found

Conditions	Consequence	Confidence
Brain Method, Long Parameter List	Long Method	1.00
Brain Method, LargeClass	Long Method	1.00
Brain Method	Long Method	0.97
Brain Method, Significant Duplication	Long Method	0.96
Brain Method	Significant Du plication	0.82
Brain Method, Long Method	Significant Duplication	0.81
Brain Method	Significant Duplication, Long Method	0.79
Large Class	Significant Duplication	0.78
Long Method	Significant Duplication	0.76
God Class	Siignificant Duplication	0.93

descriptions of the code smells. The importance of the knowledge about relations betwen code smells for the detection process has been suggested by Walter and Pietrzak [7]. They identified and exemplified several types of relations. However, there is still a need for an experimental-based, quantitative analysis on this topic.

In this subsection we present results on discovery and examination of associations between collocated smells. By *collocated smells* we mean code smells that co-exist in a single structural entity of a program: a class or a method, depending on the granularity level of the smells.

In Table 5 we report the discovered rules with highest confidence, provided that support ratio is greater than 0.03.

In most cases, the disovered rules represent the aggregate support relations (i.e. several code smells imply the presence of another smell), but there are also instances of a plain (e.g. Brain Method implies Long Method) and a mutual (or even circular) support (e.g. Significant Duplication accompanied by God Class implies Large Class presence, and Significant Duplication with Large Class implies God Class smell) .

Noticeably, a few code smells (Brain Method, Long Method and Significant Duplication) are present in most generated rules, both as condition and decision attributes. If we consider also rules with the support ratio lower than 0.03, we get even more highly-confident rules that feature the same code smells. It is evident then that they tend to go together in clique-like patterns. Most of them describe considerable violations of design principles, related mainly to the excessive complexity of the classes and methods. Therefore, their co-existence is not surprising.

Table 6. Best MP→CS rules found

System	Conditions	Consequence	Confidence
JHotDraw	Data manager, Extender	Significant Duplication	0.72
	Trait	Long Parameter List	0.83
	Outline	Significant Duplication	0.76
	Sink	Significant Duplication	0.62
GanttProject 10.0	Data manager, Extender, Immutable	Data Class	0.67
	Compound box, Function object	Data Class	0.62
GanttProject 10.1	Data manager, Extender, Immutable	Data Class	0.67
	Compound box, Function object	Data Class	0.62
GanttProject 10.2	Data manager, Extender, Immutable	Data Class	0.67
	Compound box, Function object	Data Class	0.62
GanttProject 10.3	Compound box, Function object	Data Class	0.62
GanttProject 11.1	Data manager, Immutable, Sink	Data Class	0.59

5 Micro patterns and Code Smells dependencies

There is a significant difference between code smells and micro patterns. Code smells are vague, ambiguous and subjective, whereas the micro patterns are lower-lever and can be automatically detected in an easy way. Therefore, the dependencies between micro patterns and code smells can provide different data than code smells relations, so they deserve separate investigation.

Below we present results of the analysis we performed on the relations between micro patterns and code smells. In Table 7 we report the results for GanttProject 1.10.2 (for brevity, we do not include the tables for other versions of GanttProject).

By a *match* we mean a class that contains both a smell and a micro pattern. Tables 7 and 8 report, for each code smell, the micro pattern combination and the number of smell instances in classes containing the given micro pattern.

Then we computed the percentage of the matches. In Table 8 we report the same analysis for JHotDraw. As we can notice, this project is crippled with a larger number of different smells, as in GanttProject some code smells have not been detected.

For each version of the analyzed system we considered the relations existing with higher percentage and maunally identified possible correlations, validated

Table 7. Matches of micro patterns and code smells in GanttProject 1.10.2

	BM	IC	DC	SD	GC	DaC	FE	SG	LM	LC	LPL
Overrider	0	0	0	3	0	7	0	0	2	0	7
Sink	10	18	0	12	11	24	7	3	22	9	37
Function pointer	0	0	0	0	0	0	0	0	0	0	2
Function object	0	0	0	2	1	14	0	0	0	0	7
Immutable	0	1	0	2	1	5	1	0	5	1	3
Outline	1	1	0	2	0	11	1	0	3	0	15
Data manager	0	0	0	0	0	0	0	0	1	0	1
Compound box	0	0	0	0	0	8	0	0	0	0	4
State machine	0	0	0	0	0	6	0	1	1	0	1
Record	0	0	0	0	0	0	0	0	1	0	1
Extender	0	0	0	0	0	2	0	0	0	0	1
Trait	0	0	0	0	0	0	0	0	0	0	0

Table 8. Matches of micro patterns and code smells in JHotDraw

	BM	IC	DC	SD	BC	GC	DaC	RB	MTM	SS	FE	SG	LM	LC	LPL
Function obj	1	4	0	19	0	0	0	0	1	0	1	0	3	1	6
Overrider	6	7	0	22	0	1	1	0	1	0	2	0	12	0	19
Sink	33	30	0	146	8	14	7	6	18	4	17	0	67	31	79
Outline	1	3	0	7	1	1	1	1	5	0	0	0	1	2	3
Compound box	2	2	0	7	1	2	0	0	0	0	1	0	3	2	5
Extender	0	1	0	6	0	0	5	0	1	0	0	0	2	0	4
Function ptr	1	0	0	6	0	0	0	0	0	0	1	0	3	0	2
Record	1	0	0	2	0	0	0	0	0	0	1	0	1	1	3
Immutable	8	6	0	33	1	3	5	1	0	0	7	0	15	3	30
Trait	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
State machine	0	0	0	0	0	0	0	0	2	0	0	0	0	1	2
Data manager	0	1	0	4	0	0	0	0	0	0	0	0	1	0	1

next with associative algorithms. In Table 6 we report some of the correlations we found by analyzing the codebase. For example, a couple of features {*Data manager*, *Extender*} implies the presence of a *Significant Duplication* with an accuracy of 71%.

With the performed analysis, we found that only the *Data manager*, *Extender*, *Outline*, *Compound Box*, *Function Object* and *Sink* micro patterns have significant capability of revealing the presence of three code smells, namely *Significant Duplication*, *Data Class* and *Long Parameter List*. Other correlations do exist, but their accuracy is by far lower.

6 Concluding Remarks

In this paper we analyzed possible correlations existing among code smells and between code smells and micro patterns.

The knowledge of existing relations can be useful to improve the code smell detection accuracy and improve the analysis on code quality evaluation. In particular, it is interesting to observe if some smells tend to go together or if detecting one or more micro patterns implies the presence of one or more smells. In this paper we found significant dependencies of certain smells related to the excessive size or complexity of the code. Correlations of micro patterns and smells are less apparent, but they suggest that Data Class has a relation with the Data manager, Extender, Immutable and Function Object micro patterns, while code duplication if often co-located with Data manager, Extender, Outline and Sink. It allows for stating that knowledge of the relations between code smells and micro patterns appears useful in source code assessment. However, further analysis on larger codebases is still needed to confirm these outcomes.

As this experimentation had limited scope, several improvements can be made. All the smells detected by InFusion and PMD describe exceeding the upper limit of the validity interval for a given characteristic. On the other hand, smells breaking the lower limit of the interval, which could reveal another set of closely-related smells, are not detected by those tools. It seems reasonable to extend the set of exploited detectors to enhance research and discover more complex rules.

Another possible extension is related with the of subjectivity in code smell detection with only one detector. In order to mitigate it, we could use several detectors for a single smell, with a voting mechanism used for reconciling the results. The approach should remove uncertainty of the smell presence in case of contradictions between different tools. We plan to analyze also a wider range of systems from different domains to manage other threats of validity of the study.

We are also interested in discovering other, more complex relations existing among smells and among smells and the possible combinations of micro patterns. Another research direction we consider is the analysis of the impact of refactoring on different quality metrics values, with the aim to prioritize the smells to be removed.

References

1. Kaner, C., Bond, W.P.: Software engineering metrics: What do they measure and how do we know? In: Proceedings of the 10th International Software Metrics Symposium (Metrics 2004). (September 2004) `http://www.kaner.com/pdfs/metrics2004.pdf`.
2. Riaz, M., Mendes, E., Tempero, E.: A systematic review of software maintainability prediction and metrics. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society (October 2009) 367–377 doi:10.1109/ESEM.2009.5314233.
3. Tahvildari, L., Kontogiannis, K.: A metric-based approach to enhance design quality through meta-pattern transformations. In: Proceedings of the 7th European Conference on Software Maintenance and Reengineering, IEEE Computer Society (March 2003) 183–192 doi:10.1109/CSMR.2003.1192426.

4. Lanza, M., Marinescu, R.: Object-Oriented Metrics in Practice. Springer-Verlag (2006)
5. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA (1999) `http://www.refactoring.com/`.
6. Gil, J.Y., Maman, I.: Micro patterns in java code. In: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '05), ACM (2005) 97–116 doi:10.1145/1094811.1094819.
7. Pietrzak, B., Walter, B.: Leveraging code smell detection with inter-smell relations. In Abrahamsson, P., Marchesi, M., Succi, G., eds.: Extreme Programming and Agile Processes in Software Engineering. Volume 4044 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006) 75–84
8. Mäntylä, M., Lassenius, C.: Subjective evaluation of software evolvability using code smells: An empirical study. Empirical Software Engineering **11** (2006) 395–431 doi:10.1007/s10664-006-9002-8.
9. Khomh, F., Di Penta, M., Gueheneuc, Y.G.: An exploratory study of the impact of code smells on software change-proneness. In: Proceedings of the 16th Working Conference on Reverse Engineering (WCRE '09), IEEE Computer Society (October 2009) 75–84 doi:10.1109/WCRE.2009.28.
10. Parasoft: Codewizard — now available as C++test^{TM}. Web Site (2011) `http://www.parasoft.com/jsp/products/cpptest.jsp/index.htm`.
11. Tsantalis, N., Chaikalis, T., Chatzigeorgiou, A.: Jdeodorant: Identification and removal of type-checking bad smells. In: Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR 2008). (april 2008) 329–331 doi:10.1109/CSMR.2008.4493342.
12. LOOSE Research Group: iPlasma. Web site `http://loose.upt.ro/reengineering/research/iplasma`.
13. Checkstyle: Checkstyle. Web site (2011) `http://checkstyle.sourceforge.net`.
14. Murphy-Hill, E., Black, A.P.: An interactive ambient visualization for code smells. In: Proceedings of the 5th international symposium on Software visualization (SOFTVIS '10), New York, NY, USA, ACM (2010) 5–14 doi:10.1145/1879211.1879216.
15. Arcelli Fontana, F., Zanoni, M.: A tool for design pattern detection and software architecture reconstruction. Information Sciences **181**(7) (2011) 1306–1324 doi:10.1016/j.ins.2010.12.002.
16. Software Evolution and Reverse Engineering (ESSERE) Lab: Micro structures detector. Web Site (2011) `http://essere.disco.unimib.it/reverse/Marple`.
17. Scheffer, T.: Finding association rules that trade support optimally against confidence. In De Raedt, L., Siebes, A., eds.: Principles of Data Mining and Knowledge Discovery. Volume 2168 of Lecture Notes in Computer Science. Springer Berlin /Heidelberg (2001) 424–435 doi:10.1007/3-540-44794-6_35.