Numerical Signatures of Antipatterns: An Approach based on B-Splines

Rocco Oliveto
Software Engineering Lab (SE@SA Lab)
DMI – University of Salerno, Italy
roliveto@unisa.it

Foutse Khomh, Giuliano Antoniol, and Yann-Gaël Guéhéneuc
SOCCER Lab and Ptidej Team
École Polytechnique de Montréal, QC, Canada
foutsekh@iro.umontreal.ca, {giuliano.antoniol, yann-gael.gueheneuc}@polymtl.ca

***Abstract*—Antipatterns are poor object-oriented solutions to recurring design problems. The identification of occurrences of antipatterns in systems has received recently some attention but current approaches have two main limitations: either (1) they classify classes strictly as being or not antipatterns, and thus cannot report accurate information for borderline classes, or (2) they return the probabilities of classes to be antipatterns but they require an expensive tuning by experts to have acceptable accuracy. To mitigate such limitations, we introduce a new identification approach, ABS (Antipattern identification using B-Splines), based on a numerical analysis technique. The results of a preliminary study show that ABS generally outperforms previous approaches in terms of accuracy when used to identify Blobs.**

I. Introduction

Antipatterns [1] are poor solutions to recurring design problems. They occur in object-oriented systems when developers unwillingly introduce them while designing and implementing the classes of their systems. Antipatterns have a negative impact on the quality of a system [1], [2]. Consequently, their identification has received recently more attention from both researchers and practitioners who have proposed various approaches to detect them. In particular, two approaches were proposed: DECOR, [3] the first systematic method to specify and generate automatically detection algorithms for code smells and antipatterns and Bayesian Beliefs Networks (BBNs), an approach based on BBNs for ranking classes according to their probabilities of participating in antipatterns [4].

However, previous approaches have two main limitations: either (1) they classify classes strictly as being or not antipatterns and thus cannot report accurate information for borderline classes (*e.g.*, DECOR), or (2) they return the probabilities of classes to be antipatterns but require expensive (in time and knowledge) tuning by experts to have acceptable accuracy (*e.g.*, BBNs). The first limitation lead to the "submarine" effect: several classes may be very close to be identified as antipatterns but remain under the threshold during their evolution. Minor changes can then bring them all above the threshold, falsely leading developers to suspect the latest changes as culprit. The second limitation cause the detection results to rely too much on the experts' judgement. An incomplete experts' knowledge can cause a high number of false positives, resulting in a waste of time and resources for developers and managers that must skim through the results. Moreover, a model built in a given context is not generalisable to other contexts and should be recalibrated to be effective, which is a difficult task as historical data and context knowledge are not always available.

In this paper, we propose a new identification approach, called ABS (Antipattern identification using B-Splines), to help overcome these limitations. ABS is based on a learning technique using an interpolation method from the numerical analysis field. The basis of ABS is the building of *signatures* of classes based on quality metrics, as in [5], using B-splines [6] to abstract the metric values. ABS models a class using specific interpolation curves (*i.e.*, B-splines) of plots mapping metrics and their values for the class. The similarity of a given class to an antipattern is computed by calculating the distance between the curve of the class and the curves of classes previously classified as antipatterns (or good classes). Like BBNs, ABS needs a corpus of antipatterns. However, in contrary to BBNs, ABS does not need experts' knowledge to define a learning structure; thus, it reduces the bias introduced in BBNs by the experts' subjectivity when structuring the BBNs of the antipatterns.

We have implemented ABS in "Sign-o-meter", a tool to assist developers in assessing quickly the probability of classes to become Blobs and the evolution of this probability. Moreover, we performed a preliminary study applying ABS to detect the Blob antipattern. The identification accuracy of ABS was compared with DECOR and BBNs. The case study shows that, in general, ABS provides better results than BBNs and DECOR. In few explainable cases, ABS has lower accuracy than previous approaches, *i.e.*, when the training set to build the signature is too small.

The paper is organised as follows. Section II discusses previous work. Section III describes the proposed approach for identifying antipatterns, while Section IV presents the results of a preliminary study carried out to evaluate the proposed approach and concludes the paper.

II. Related Work

Webster [7] wrote the first book on "antipatterns" in object-oriented development; his contribution covers conceptual, political, coding, and quality-assurance problems.

Riel [8] defined 61 heuristics characterising good object-oriented programming to assess software quality manually and improve design and implementation. Beck [2] defined 22 code smells, suggesting where developers should apply refactorings. Mäntylä [9] and Wake [10] proposed classifications of code smells. Brown *et al.* [1] described 40 antipatterns, including the well-known Blob.

Several approaches to specify and identify code smells and antipatterns have been proposed in the literature. They range from manual approaches, based on inspection techniques [11], to metric-based heuristics [3], [12], [13], where antipatterns are identified according to sets of rules and thresholds defined on various metrics. In [12] the detection of smells is based on deviations from good design principles and consist of combining metrics with set operators and comparing their values against absolute and relative thresholds. In [13] heuristics to identify code smells are derived from template similar to the one used for design patterns. Moha *et al.* [3] proposed the DECOR method to specify and automatically generate identification algorithms. DECOR provides identification algorithm with good precision and perfect recall while allowing quality analysts to adapt the specifications to their context. It is the current state-of-the-art threshold-based identification approaches.

Khomh *et al.* [4] argued that threshold-based approaches do not handle the uncertainty of the detection results and, therefore, miss borderline classes. Consequently, they proposed a BBN for the identification of Blobs in systems, which output is the probability that a class exhibiting the characteristics of Blob is truly a Blob. Similar to our approach, a BBN is able to qualify continuously the probability of classes to be antipatterns.

Some visualisation techniques, for example [14], were used to find a compromise between fully-automatic identification techniques and manual inspections. Other approaches perform fully-automatic identification and use visualisation to present the identification results [15], [16].

Other related approaches include architectural consistency checkers, which have been integrated in style-oriented architectural development environments [17], [18], [19]. For example, active agents acting as critics [19] can check properties of architectural descriptions and identify potential syntactic and semantic errors.

III. Antipattern Identification using B-splines

This section presents our approach, ABS (Antipattern identification using B-Splines), for the identification of antipatterns using their signatures, described as B-splines [6]. Recently, B-splines were used to characterise software artefacts (documentation and code) to compute their textual similarity for traceability recovery [20].

ABS first models each class by its particular interpolation curves, *i.e.*, its B-splines, built using a set of metrics and their values for the class. In ABS, we also model antipatterns

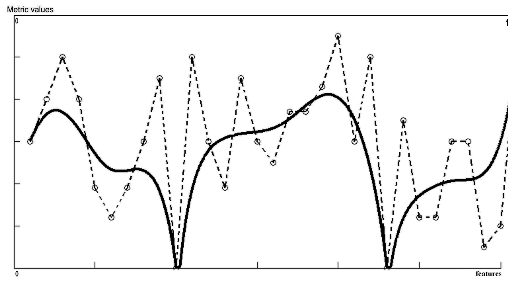

Figure 1. Class/antipatterns representation using B-splines

using B-splines, inferred from a set of classes known as participating in the antipatterns. Then, we estimate the risk of a class to be an antipattern by computing the similarity of its signature from the signature of known antipattern. To improve accuracy when computing the risk, we also consider the distance from both antipatterns and a set of good quality classes. We define good classes as any classes that does not participate in an antipattern. The similarity is then computed by calculating the similarity between the corresponding interpolation curves.

A. Identifying the Signature of Classes

A class can be described by a set of metric values; metrics measuring for example the class cohesion, coupling, and complexity. Let $M = m_1, m_2, \ldots, m_n$ be a set of metrics computed for each class of a system, then a class c_j is represented by the set of points $\{p_1, p_2, \ldots, p_n\}$ where the X and Y coordinates of the generic point p_i are i and $m_{i,j}$.

Such a representation allows using a numerical analysis technique to define the signature of the class by interpolating the points associated to the class. We consider the points representing a class as control points of a uniform B-spline curve [6], which is a generalisation of a Bézier curve, *i.e.*, a piece-wise Bézier curves [6].

Figure 1 shows the graphical representation of a class. The dash and the bold lines denote the control polynomial and the B-spline curve representing the class, respectively. The B-spline is a curve approximation technique where the control points influence the shape of the curve but the curve does not interpolate the control points, except for the first and last points [6]. However, it is possible to force the B-spline to interpolate a set of given points [6]. We force the B-spline to interpolate the points representing metrics with values equal to 0, as shown in Figure 1: in this way we do not give to these points the average value of the preceding and following points.

We choose a B-spline to derive a curve representing a class because (i) it is fast and easy to compute, (ii) it provides local control on the curve, and (iii) the degree of the B-spline is independent of the number of control points. In particular, we empirically observed that computing the B-spline never took more than 1 second and that $k = 30$ is

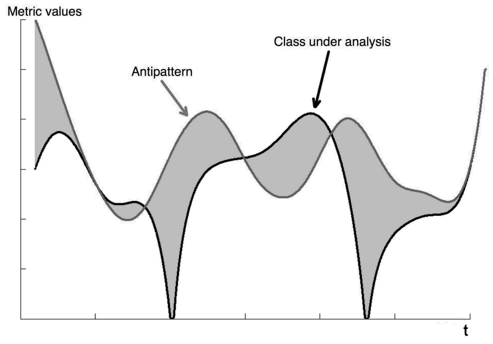

Figure 2. Signature comparison using B-splines

a good compromise between computational complexity and identification accuracy. The latter observation is consistent with that in previous work [20].

B. Comparing the Signature of Classes

Once obtained the signatures of each class of a system, it is possible to compare them with those of classes previously classified as antipatterns or good quality classes. The conjecture is that if a class has a signature similar to that of a previously classified antipattern, *e.g.*, God class [8], and different from a set of classes previously classified as good quality classes, then the class under analysis is probably the considered antipattern, *e.g.*, a God class.

Let $A = \{a_1, a_2, \ldots, a_p\}$ be the set of classes previously classified as being an antipattern[1], and $G = \{g_1, g_2, \ldots, g_q\}$ be the set of classes previously classified as good classes. We compute the similarity between the signature of c_i and the signature of every elements in A and G and rank—in a decreasing order—each pair according to these similarities. Thus, the risk that a generic class c_i is an antipattern is:

$$\begin{aligned} godliness(c_i, A, G) \quad = \quad & \sum_{j=1}^{p} w_{A_{i,j}} \cdot similarity(c_i, a_j) - \\ & \sum_{k=1}^{q} w_{G_{i,k}} \cdot similarity(c_i, g_k) \end{aligned}$$

where $w_{A_{i,j}} = \frac{1}{pos(c_i, a_j)}$ and $w_{G_{i,k}} = \frac{1}{pos(c_i, g_k)}$ represent weights based on the positions of the pairs (c_i, a_j) and (c_i, g_k) in the ranked list. The higher the "godliness" [4] value, the higher the likelihood that c_i is an antipattern.

The similarity between the class signatures of class c_i and of antipattern a_j—or good quality class g_k—is based on the distance between the corresponding B-spline curves:

$$similarity(c_i, a_j) = 1 - D(c_i, a_j)$$

where $D(c_i, a_j) \in [0, 1]$ represents the distance between the B-splines of c_i and a_j and is calculated using the normalised 1-norm (see Figure 2).

[1]These classes can be class of a same system or classes taken from different systems.

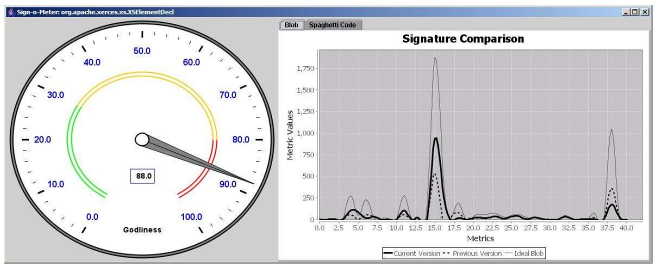

Figure 3. Screen-shot of Sign-o-Meter

The order of the metrics on the X-axis influences the representation of a class and the distance between two classes (or a class and an antipattern). However, we studied different ways of ordering the metrics and did not find any substantial difference in the identification results.

C. Visualising and Following the Evolution of Signatures

The graphical representation of the B-splines characterising classes and antipatterns is also useful to monitor the evolution of classes in time. We implemented a tool, called Sign-o-Meter, that displays two related and complementary views of the signatures of a class:

1) One dial showing the similarity between the class under development and an antipattern using the godliness metric scaled in $[0, 100]$.
2) A plot of three B-spline curves representing, respectively, the current signature of the class, the signature of the previous version of the class, and the signature of an ideal antipattern, defined as the average signature of the previously classified antipatterns.

Figure 3 illustrates a usage scenario of Sign-o-Meter, where the tool is used to identify Blobs (or God classes). On the left hand side, it shows the dial displaying the godliness measure of the class. Higher is the value, higher is the likelihood that the class is a Blob. On the right hand side, it shows a dash B-Spline curve for the previous version of the class, a thin continuous line for the B-spline curve of the current version of the class, and a thick line for the curve characterising the ideal Blob (from the previous classified Blob). Such a plot provides information on the evolution of the class by allowing developers to compare the signatures of the current version of a class to the one of its previous version and to the ideal antipattern. Consequently, it allows developers to realise that their class is moving towards or away from the ideal antipattern and, thus, to take the appropriate actions.

IV. Conclusion and Future Work

We presented a novel approach, ABS, to identify occurrences of antipatterns using the signature of the classes and of the antipatterns. The signature of a class (or antipattern) is represented by a particular interpolation curve, *i.e.*, its

B-spline, built using a set of metrics and their values for the class. Then, we estimate the risk of a class to be an antipattern by computing the similarity of its signature with the signature of known antipattern and the distance from a set of good quality classes.

A preliminary evaluation of ABS was conducted to identify the Blob antipattern. The accuracy of ABS was compared with DECOR [3], which uses strict thresholds, and an approach based on BBNs [4]. The case study was conducted on two medium size open-source Java systems. We studied the accuracy of ABS in the two following scenarios: (i) intra-system identification, where we assumed that historical data (*i.e.*, correctly identified Blobs) are available for a given system and where we used this data to identify other Blobs in the same system; and (ii) extra-system identification, where we assumed that a quality analyst has access to the historical data from one system and uses it to identify Blobs in the other system. We found that ABS outperforms in general previous approaches in precision and recall while being more attractive in practice thanks to its speed and ease to generate and interpret the signatures. In few explainable cases, ABS has lower precision or recall than previous approaches, when the training set to build the signature is too small.

Future work include replicating our cast study on larger systems to assess the generalisability of our novel approach. It also include computing the signatures of other antipatterns than the Blob and again replicate our study to assess the impact of the antipattern on accuracy.

Acknowledgment

This work has been partly funded by the Canada Research Chair on Software Patterns and Patterns of Software.

References

[1] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, 1^{st} ed. John Wiley and Sons, 1998.

[2] M. Fowler, *Refactoring – Improving the Design of Existing Code*, 1^{st} ed. Addison-Wesley, 1999.

[3] N. Moha, Y. G. Guéhéneuc, L. Duchien, and A. F. L. Meur, "DECOR: A method for the specification and detection of code and design smells," *Transactions on Software Engineering*, 2009.

[4] F. Khomh, S. Vaucher, Y. G. Guéhéneuc, and H. Sahraoui, "A bayesian approach for the detection of code and design smells," in *Proceedings of the* 9^{th} *International Conference on Quality Software*, IEEE CS Press, 2009.

[5] Y. G. Guéhéneuc, H. Sahraoui, and F. Zaidi, "Fingerprinting design patterns," in *Proceedings of the* 11^{th} *Working Conference on Reverse Engineering*, IEEE CS Press, 2004, pp. 172–181.

[6] C. de Boor, "On calculating with b-splines," *Journal of Approximation Theory*, vol. 6, pp. 50–62, 1972.

[7] B. F. Webster, *Pitfalls of Object Oriented Development*, 1^{st} ed. M & T Books, 1995.

[8] A. J. Riel, *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.

[9] M. Mantyla, "Bad smells in software - a taxonomy and an empirical study," Ph.D. dissertation, Helsinki University of Technology, 2003.

[10] W. C. Wake, *Refactoring Workbook*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[11] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," in *Proceedings of the* 14^{th} *Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM Press, 1999, pp. 47–56.

[12] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *Proceedings of the* 20^{th} *International Conference on Software Maintenance*. IEEE CS Press, 2004, pp. 350–359.

[13] M. J. Munro, "Product metrics for automatic identification of "bad smell" design problems in java source-code," in *Proceedings of the* 11^{th} *International Software Metrics Symposium*, IEEE Computer Society Press, 2005.

[14] F. Simon, F. Steinbrückner, and C. Lewerentz, "Metrics based refactoring," in *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*. IEEE CS Press, 2001, p. 30.

[15] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.

[16] E. van Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Proceedings of the 9th Working Conference on Reverse Engineering*. IEEE CS Press, 2002.

[17] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is so hard," *IEEE Software*, vol. 12, no. 6, pp. 17–26, 1995.

[18] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213–249, 1997.

[19] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, "A comprehensive approach for the development of modular software architecture description languages," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 2, pp. 199–245, 2005.

[20] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Traceability recovery using numerical analysis," in *Proceedings of 16th Working Conference on Reverse Engineering*. IEEE CS Press, 2009, pp. 195–204.