PODPIS ZAUFANY
JERZY
NAWROCKI
21.02.2023 06:35:00 [GMT+1]
Dokument podpisany elektronicznie podpisem zaufanym

Prof. dr hab. inż. Jerzy Nawrocki
Instytut Informatyki
Politechnika Poznańska

Poznań, 2023-02-19

Review of PhD dissertation by

Łukasz Puławski

entitled:

Methods of detecting spatio-temporal patterns in software development processes

1. Introduction

Anti-patterns and code smells are problematic areas within source code. Many authors, including the author of the dissertation, use those terms interchangeably, and in this review I will do the same. It is justifiable as both of them describe some properties of a code fragment and both of them have a detrimental effect on software development. According to a recently published research (see e.g. [Mo-2021]), anti-patterns – if they develop in a code fragment – can lead to significantly greater bug frequency and change frequency compared to code fragments that are free of them. So, they should be avoided and there are various methods and tools aimed at identifying them in code but the problem has not been solved yet. The author has decided to contribute to this area as well by trying to answer the following research questions (formulation of the questions is mine):

RQ1. How to predict occurrence of anti-patterns?

If anti-patterns lead to troubles, it would be worth to know as early as possible where they can appear.

RQ2. How to make the method of anti-pattern prediction time efficient?

People do not like to wait, so the faster prediction process the better (i.e. more frequently it will be used and there is a chance that some countermeasures, e.g. code refactoring, will be taken earlier).

RQ3. What is effectiveness of the proposed approach?

In other words, how good is the proposed approach in predicting appearance of anti-patterns. To answer this question experimental validation has been conducted.

The dissertation is oriented towards software written in Java, which is one of the most popular programming languages, and it is assumed that change management is based on Subversion (also known as SVN) or Git, two very popular version control tools.

The above questions constitute a research challenge and they are also important from practical point of view.

As it comes to the text, the dissertation consists of 7 chapters. The most important is Chapter 6 where the reader will find mathematical theory of predicting anti-patterns (it has over 80 pages) and Appendix A which contains results of experimental validation conducted by the author. The dissertation is over 250-pages long (!) and I consider this a weakness.

2. Prediction of occurrence of anti-patterns

Strength 1. *The idea of predicting occurrences of anti-patterns in a given source code seems original.*

Explanation. The author first presents means for describing **spatial relation** between pattern instances in a source code, and then extends them with **temporal relations**. This is a basis for creating **decision tables** out of training code repositories. Having a decision table one can use the C4.5 algorithm (developed by Ross Quinlan) to get a **decision tree**. The decision tree (or its rule-based representation) can be used to identify fragments in the source code where anti-patterns can appear in the future, so a countermeasure (known as code refactoring) can be taken early.

Weakness 1. *Description of spatial relation between anti-pattern instances is based on graph-theoretic approach. Unfortunately, there is **lack of discussion on similar work done before**.*

Explanation. The central concept concerning spatial relation is software snapshot. The author first introduced the notion of dependency relation (Definition 19, pp. 94-95) which is a foundation for his **graph-based model** of a software version. Then he defined software snapshot, SSn_r, which corresponds to a single version of a software and is represented as a triple consisting of (see p. 96):

- a set of code entities (e.g. classes, methods etc.),
- dependences between them (e.g. *call* or *contain* relations), and
- metrics (e.g. lines of code or cyclomatic complexity).

This approach has been further developed through definitions 26, 28, 30, 32 – 37 (incl. equations 6.6 – 6.8). Anti-pattern instance is a subgraph in software snapshot.

Unfortunately, some **important papers presenting related work have been ignored** by the author:

- In [Mens-2005] a similar idea of **graph representation of source code** was presented in the context of code refactorization. Using the terminology from [Mens-2005], a pattern instance resembles very much 'graph expression' – the only thing that should be added is set of metrics.
- In [Strein-2007] an **extensible meta-model for code analysis** is presented. The work seems very important in the context of the dissertation. The meta-model presented there builds on the Dagstuhl Metamodel and has some features not present in the snapshot metamodel proposed in the dissertation (it has a 3-layer structure with a language-specific model, a common model which is language independent, and a view model which is tailored to a specific analysis; moreover abstract syntax tree representation of source files which important but only mentioned on p. 99 is directly included into the extensible meta-model proposed by Strein *et al*.). It seems the dissertation could benefit from the ideas presented in this paper. Several examples of using the concepts presented in [Strein-2007] can be found on the web pages of ARiSA company [ARISA-2009].

Strength 2. *The idea of **2-phase creation of decision tables** corresponding to anti-patterns is **ingenious** .*

Explanation. The idea of 2-phase creation of decision tables is described on pp. 159-161. An occurrence of a pattern instance, x_j, is represented in a decision table as:

- a feature vector $A(x_j)$ describing the 'situation' corresponding to this occurrence, and
- a decision attribute which, in general, would contain pattern names T or *NOT*_T values.

For the sake of understandability let us consider a single anti-pattern. Then the question associated with the decision column could be:

has the pattern occurred in the situation described by feature vector $A(x_j)$?

and decision attributes would assume binary values *Yes* or *No* (*Yes* has replaced pattern name T and *No* corresponds to *NOT*_T value). As the process is focused on **finding** instances of the pattern in the code, the decision column of the table contains only *Yes* values, so it would be extremally unbalanced. To cope with this the author introduced the **second phase** in which the snapshot representation of the code repository is searched for 'situations' corresponding to $A(x_j)$ **without** an instance of the pattern (in the dissertation it is called *random equivalent*) and then a row with feature vector A(x_j) and decision attribute *No* is added (according to the author's proposition, candidate situations for *No* answer are chosen randomly).

∎

On page 49 the author declares (bolding has been added by me):

> *we use* ***rule-based classifiers*** *to find spatio-temporal patterns in the software development process and express these patters directly in terms of the software engineering domain*.

In this context the following weakness is really surprising:

Weakness 2. The dissertation **does not contain** any **decision table**, **decision tree** nor prediction rule presenting the knowledge acquired during the validation experiment what makes quality assessment practically impossible.

Explanation. As one of the goals of the dissertation was to propose a method of predicting occurrences of anti-patterns in a form of a rule-based classifier it would be very much needed to see the prediction rules (or decision trees produced by C4.5) corresponding to the anti-patterns described in Sec. 6.4. As the basis for decision tree generation is a decision table, I expected that some decision tables will be also presented but they are not.
To avoid flood of data, the author could choose one repository (e.g. ElasticSearch – from Table A.11 on p. 236 it follows that this repository is the richest one in the sense of occurrence of anti-patterns) and present single-pattern-oriented tables for the patterns discussed in Sec. 6.4 (or at least for some of them). Exemplary decision tables could provide important information about the effectiveness of the proposed approach. Without them it is very hard to say if the proposed method of creating decision tables works correctly.

∎

Weakness 3. *There is* ***lack of analysis of applicability of Allen's relations****. In the context of anti-pattern* ***prediction*** *some of them seem superfluous and should be omitted (it would have also a practical implication as the number of columns of decision tables would become smaller)*.

Explanation. In Section 4.3.1 (p. 60) the author presents six Allen's relations (so called *non-inverted* relations – see Fig. 1). They are later used to describe possible time relations between occurrences of anti-pattern instances and are also part of labels assigned to columns of decision tables (see Example 2 on p. 157). The problem is that **in the context of prediction some of them seem to have no sense** and could be neglected. As the author has noticed, in the case of commit-driven software evolution time is discrete, not continuous. Therefore, each interval of code revisions is a closed interval. Let us consider two closed intervals:

[x1, x2] corresponding to an anti-pattern instance π1 playing the role of trigger, and
[y1, y2] corresponding to an anti-pattern instance π2 being triggered by the occurrence of π1,

1. X *takes place before* Y ($\exists s \in \Re : x_2 < s < y_1$)
2. X *meets* Y ($x_2 = y_1$)
3. X *overlaps* Y ($x_1 < y_1 < x_2 < y_2$)
4. X *starts* Y ($x_1 = y_1 \wedge x_2 < y_2$)
5. X *contains* Y ($x_1 < y_1 < y_2 < x_2$)
6. X *is finished by* Y ($x_1 < y_1 \wedge y_2 = x_2$)

Fig. 1. Six Allen's relations used in the dissertation.

The following Allen's relation

4. X *starts* Y (x1 = y1 ∧ x2 < y2)

seems useless in the context of predicting occurrence of π2. It is so, because it means that π2 shows up at the same moment as π1, so there is no chance for predicting occurrence of π2.

Moreover, the following Allen's relations

2. X *meets* Y (x2 = y1)
5. X *contains* Y (x1 < y1 < y2 < x2)
6. X *is finished by* Y (x1 < y1 ∧ y2 = x2)

are equivalent to the *overlaps* relation (from the point of view of π2 prediction). Relation 2 is a special case of overlaps (the overlap is just 1 revision long), and relations 5 and 6 say exactly the same as relation 3: when π1 started and still existed then after some time π2 occurred.

The conclusion is that from the point of view of anti-pattern prediction **two Allen's relations should be enough: *before*** (No. 1) **and *overlaps*** (No. 3).

■

Weakness 4. *It is not clear* ***what is really discovered*** *in the source code by the apparatus developed in the dissertation:* ***rules of prediction*** *or* ***rules of co-existence*** *of anti-patterns. This is an important issue which is not discussed in the dissertation*.

Explanation. As B. Pietrzak and B. Walter have noticed long time ago, some anti-patterns have a tendency to co-exist [Pietrzak-2006]. This is especially the case of the *starts* relation (see Fig. 1) but not only. If two anti-patterns, π1 and π2, are in the co-existence relation then sometimes π1 will precede π2 and sometimes it will be the other way round. In the context of [Pietrzak-2006] it would be interesting to see if there are real opportunities for anti-pattern prediction or what we have is just anti-pattern co-existence. To benefit from anti-pattern prediction in source code one needs time to react (i.e. to conduct code refactorization to remove them), so anti-pattern prediction should be discussed in terms of **time distance**: how many code revisions (commits) separate occurrence of anti-pattern π1 from occurrence of anti-pattern π2. The larger the distance the more chances to eliminate a triggering anti-pattern. And the other way round: if time distance is very short, in practical terms it means that what one has to cope with is anti-pattern co-existence and anti-pattern prediction tools become of little importance ('traditional' anti-pattern detection would do).

■

Weakness 5. *Mining of spatio-temporal rules (Chapter 6) is not clearly presented.*

Explanation. In Chapter 6 four aspects are discussed:

- construction of *decision tables* from code repositories,
- so called *adaptive approach* aimed at saving computation time,
- *pattern mining* (i.e. detecting anti-patterns in the source code),
- *experimental validation* of the proposed solutions.

The first problem is that the above threads of thinking are **interleaving** and scattered over sections of Chapter 6 what hinders understanding (see Table 1). With each of those aspects there is a separate set of definitions and symbols the reader is to remember, so it would be better if the author discussed those aspects one by one.

The second problem is very complicated terminology and notation. For instance, in Definition 40 (p. 144) the author introduces the following symbol:

$$Occ^{close}_{[l_1^{PI_1},A]}(P_2)$$

Moreover, what is called in Definition 46 (p. 154) a **spatio-temporal rule** and denoted as

$$(T_1, A_1, s_1) \wedge (T_2, A_2, s_2) \wedge \ldots \wedge (T_n, A_n, s_n) \rightarrow T_d,$$

is in fact a template for representing **a single object** in a decision table (unfortunately, the rules resulting from the validation experiment are in presented in the dissertation). As the value of (T_i, A_i, s_i) is a **set** (it is specified in Definition 43, p. 152), using logical operator ∧ seems not appropriate.

Assume one is interested in predicting occurrences of just one pattern P (say Anaemic Entity). Then the decision table should describe situations when pattern P occurred and when it did not. It means the decision variable d will assume just two values: True, False. The case for d = True seems easy to interpret when following the Example 2 (p. 157): if the column of decision table corresponding to tuple (T_i, A_i, s_i) has in a given row value k_i it means there are k_i instances of pattern T_i and each of them satisfies two condition:

- it is in the spatial relation s_i with the instance of pattern P, and
- its lifespan is in time relation A_i with the lifespan of the instance of P.

The following question arises: How to interpret the row with d = False? It means that one is considering a situation when there is no instance of pattern P. What is the spatial distance of an instance of pattern T_i to non-existent instance of pattern P?

▮

Tables 1. Scattering of discussion over sections of Chapter 6

Decision tables	Adaptive approach	Pattern detection	Empirical validation
Sec. 6.1			
Sec. 6.2.1*			
Sec. 6.2.2			
	Sec. 6.2.3		
		Sec. 6.3	
			Sec. 6.4
Sec. 6.5.1 – 6.5.5			
	Sec. 6.5.6 – 6.5.7		
Sec. 6.6.1 – 6.6.3			
	Sec. 6.6.4		
			Sec. 6.6.7

3. Saving execution time by means of incremental approach

Strength 3. *The author proposed incremental algorithms for computing subsequent versions of software snapshots, and updating spatial and temporal relations. This is important from practical point of view and the presented analysis is non-trivial.*

Comment. In the dissertation the word used is *adaptive*. I prefer *incremental* as what the author is doing strongly resembles a kind of memoization: instead of computing some representation from scratch for a new version of source code, the author just updates necessary parts of representation. Incremental approach source code modelling was also presented in [Strein-2007] but they focused on other type of information, however a reference to this publication would be very appropriate.

▮

4. Experimental validation

To conduct validation experiments the author has selected 8 anti-patterns (see Sec. 6.4.1 – 6.4.7) and used 7 popular open-source projects with long history of changes (see Sec. 7.1).

Strength 4. *Many assumptions made by the author about source code have been checked via experiments (see Appendix A).*

Strength 5. *From Table 6.3 (p. 169) it follows that the proposed approach is really effective in predicting occurrences of anti-patterns.*

Weakness 6. *The goals of experiments have not been clearly stated and there is lack of analysis of validity threads.*

Explanation. It is quite popular to formulate experiment goals with the GQM template[1]. I have also some doubts concerning the **definition of anti-patterns** – I guess the definitions presented in Sec. 6.4 have been used. Unfortunately, they are **non-standard** (for instance the definitions of God class and Brain class are different from those presented in [Olbrich-2010]) and because **expert tagging** described in Appendix B.1 has been **re-used** from other experiments there is a risk of lack of consistency, because those experts could use different (more popular) definition of anti-patterns.

∎

5. Other remarks

Strength 6. *From the linguistic point of view the dissertation is very well written.*

Weakness 7. *There is lack of Systematic Literature Review and some important papers have been missed by the author.*

Explanation. Here are some references that seem important but missing in the dissertation: [ARISA-2009], [Mens-2005], [Mo-2021], [Pietrzak-2006], [Sharma-2018], [Strein-2007], [Yamashita-2015].

∎

Weakness 8. There are some not so much important defects and typos that hinder understanding.

Explanation. Here are some examples.
Example 1. I do not understand the following condition which appears in Definition 46 (p. 155):

... there exists a pair (E, I) [...] such that PI is 0-overlapping-close to E ...

According to the equalities (6.9), (6.10) $Ov^{evol}(PI_1, PI_2) \geq 0$, so taking into account Definition 32 (which says that two pattern instances PI_1 and PI_2 are k-overlapping-close iff $Ov^{evol}(PI_1, PI_2) \geq k$) one can conclude that any two pattern instances PI_1, PI_2 are 0-overlapping-close, so the above condition is a tautology and should be removed.

Example 2. It seems the definition of d-distance closeness (Definition 32, p. 141) contains a typo. It reads as follows:

$$PI_1 \ and \ PI_2 \ are \ \text{d-distance-close} \ \textit{iff} \ d^{evol}(PI_1, PI_2, T) \geqslant d$$

As $d(PI_1, PI_2, T)$ is defined as minimal value of shortest distance (p. 139) and taking into account equation (6.8) defining d^{evol} it seems the condition after '*iff*' should be:

$$d^{evol}(PI_1, PI_2, T) \leq d.$$

[1] See e.g. https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/9e34301f-4194-4379-b020-b737c43d0ac2/content , Sec. 2.1.1.

Otherwise it is in conflict with Fact 19 which says that if two pattern instances are d-distance-close then they are also x-distance-close for any valid x > d. Assume two pattern instance overlap with their entities. Then from Fact 15 it follows that their evolution-wide distance is 0 and they are not x-distance close for any x > 0.

Example 3. Characteristic of a classifier of anaemic class (p. 129) seems inaccurate and Fact 5 should be corrected accordingly. Here is the definition of an anaemic class (p. 129):

1. *it has more than 8 fields,*
2. *it has more than 8 methods,*
3. *all methods but one are trivial or effectively trivial,*
4. *there are no complex constructors contained in c,*
5. *all subclasses of c satisfy the above four conditions.*

The author declares the following (also p. 129):

A classifier that does not check the 5 condition is clearly {(contain)}-bounded.

The problem is that when checking if a given method is **effectively trivial** one needs to check if the type of the method's parameter is the same as the type of the field the method refers to (see p. 128-129). The situation is presented on Fig. 2. To check if a method of a class is effectively trivial one needs to check if the only parameter of the method is of the same type as the field of the class the method refers to. It means that classifier is not {(*contain*)}-bounded because (*contain*) path is to short to reach the grey class which is expected to be the same for the '*type*' and '*parameter*' edge.

Moreover, Hypothesis H1 seems not clearly formulated. It reads as follows:

H1: There are statistically significant temporal patterns in software development process that can be used to predict the appearance of anti-patterns

Unfortunately, I have not found in the dissertation the definition *of statistically significant temporal pattern*.

■

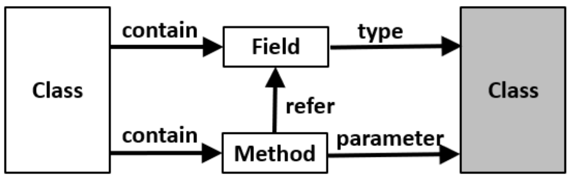

Fig. 2. A fragment of software snapshot SSn corresponding to an effectively trivial method.

6. Conclusion

I have some doubts if the dissertation is good enough to confer its author the Ph.D. degree. But on the other hand I am aware that maybe some of my negative opinions and doubts result from my lack of understanding and not clear presentation of the ideas by the author. Therefore, **I suggest to admit the author to public defence of the dissertation**.

References

[ARISA-2009] https://www.arisa.se/compendium/node88.html

[Mens-2005] Mens, T., Van Eetvelde, N., Demeyer, S., & Janssens, D. (2005). Formalizing refactorings with graph transformations. Journal of Software Maintenance and Evolution: Research and Practice, 17(4), 247-276, https://doi.org/10.1002/smr.316

[Mo-2021] R. Mo, Y. Cai, R. Kazman, L. Xiao and Q. Feng, "Architecture Anti-Patterns: Automatically Detectable Violations of Design Principles," in IEEE Transactions on Software Engineering, vol. 47, no. 5, pp. 1008-1028, 1 May 2021, doi: 10.1109/TSE.2019.2910856. https://ieeexplore.ieee.org/document/8691586

[Moha-2010] N. Moha, Y. -G. Gueheneuc, L. Duchien and A. -F. Le Meur, "DECOR: A Method for the Specification and Detection of Code and Design Smells," in IEEE Transactions on Software Engineering, vol. 36, no. 1, pp. 20-36, Jan.-Feb. 2010, doi: 10.1109/TSE.2009.50.

[Olbrich-2010] S. M. Olbrich, D. S. Cruzes and D. I. K. Sjøberg, "Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems," 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 2010, pp. 1-10, doi: 10.1109/ICSM.2010.5609564. https://ieeexplore.ieee.org/document/5609564

[Palomba-2015] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk and A. De Lucia, "Mining Version Histories for Detecting Code Smells," in *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 462-489, 1 May 2015, doi: 10.1109/TSE.2014.2372760.

[Pietrzak-2006] Pietrzak, B., Walter, B. (2006). Leveraging Code Smell Detection with Inter-smell Relations. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds) Extreme Programming and Agile Processes in Software Engineering. XP 2006. Lecture Notes in Computer Science, vol 4044. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11774129_8

[Sharma-2018] Tushar Sharma, Diomidis Spinellis, A survey on software smells, Journal of Systems and Software, Volume 138, 2018, Pages 158-173, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2017.12.034. (https://www.sciencedirect.com/science/article/pii/S0164121217303114)

[Strein-2007] Strein, D., Lincke, R., Lundberg, J., & Löwe, W. (2007). An extensible meta-model for program analysis. IEEE Transactions on Software Engineering, 33(9), 592-607.

[Yamashita-2015] A. Yamashita, M. Zanoni, F. A. Fontana and B. Walter, "Inter-smell relations in industrial and open source systems: A replication and comparative analysis," 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, Germany, 2015, pp. 121-130, doi: 10.1109/ICSM.2015.7332458.