

Relative Thresholds: Case Study to Incorporate Metrics in the Detection of Bad Smells

Yania Crespo¹, Carlos López², and Raúl Marticorena²

¹ University of Valladolid

Department of Computer Science, Valladolid (Spain)

`yania@infor.uva.es`

² University of Burgos

Area of Languages and Computer Systems, Burgos (Spain)

`{clopezno, rmartico}@ubu.es`

Abstract. To detect flaws, bad smells, etc, we often use quantitative methods: metrics or measures. It is common in practice to use thresholds to set the correctness of the measures. Most of the current tools use generic values. Nevertheless, there is a certain concern about the effects of threshold applications on obtained values.

Current research is working on case studies of thresholds for several products and different versions. However, product domain and size could also affect the results, so we deal with the question of using generic vs. relative thresholds, looking at what effects this could have in bad smell detection.

Key Words: thresholds, metrics, flaw design, bad smells, refactoring

1 Initial Context

In previous works [3, 11], we tackled the use of frameworks in order to give complete support in the refactoring process: to parse the code to a metamodel, to collect metrics from a metamodel, to detect bad smells from metrics, to support refactorings and to regenerate the transformed code.

However, in this process, there are many points not yet covered. In particular, relations between metrics and bad smells were defined on the basis of generic thresholds in [3]. Although thresholds could be customized, it would be a task for the product manager. Furthermore, the use of thresholds raises certain questions: Which values should we use? Are they correct? Are they suitable for the current product?

This paper proposes a case study of several products of medium size, also focusing on the evolution of some of them. The results obtained should make clear the need to support the relative product values to be applied, in our case, with the aim of a future integration with refactorings.

The remainder of this work is organized as follows: Section 2 shows the current state of the art, Section 3 develops the case study focusing on several products/projects in several versions. Section 4 proposes the application of the

results to bad smells detection. Section 5 concludes by showing some conclusions of the proposed solution.

2 Related Works

Most of the current environments include metric collection. They also include the possibility to fix thresholds on these metrics. It is the programmer who establishes these values, using the development guides of his/her company. These filters are, however, fixed for all products and results obtained do not always suggest catalogued flaws.

There are works in the detection of design flaws. In [9,10] Marinescu proposes the concept of design strategies. Strategies are defined on the basis of metrics and are applied to the information collected in a metamodel. The metamodel contains code information, but it is designed and implemented to allow queries (all operations are translated to SQL sentences), but this does not work for a complete refactoring process. In this work, the use of generic and relative thresholds is discussed, but their suitability is not mentioned.

In [7, 8], we find a catalogue of flaws named bad smells. In order to link them to a metric suite, this assignment suggests selecting generic thresholds in most cases. They also point out the problem of leaving these decisions to the subjective human intuition.

On the other hand, in [12], Tourwé and Mens propose the detection of refactoring opportunities using queries on a logic meta-programming environment. They define queries to suggest the corrective actions to accomplish. Following that work Muñoz, in [1], uses a set of logic queries that compute object-oriented metrics to detect these bad smells with generic thresholds.

Thresholds have also been tackled in [5]. French proposes a system based on statistical methods to determine thresholds for different products, without applying them to bad smell inference or trying to see the effects on several versions.

From these previous works, we want to provide answers to certain issues:

- the correctness of using generic vs. relative product thresholds to detect bad smells in code.
- the influence of the kind and size of software products (frameworks vs. libraries).
- the suitability of the solution on different versions of the same product.

3 Case Study

3.1 First Phase: Compararison between Products

We make a comparative study of six products. We take different products, most of them stable versions, used over a long period of time, with medium or large size. We prefer these samples instead of using “toy” samples of small/tiny size, with low functionality.

The selected products are:

- jfreechart-1.0.0.pre2 (629 classes)
- jhotdraw-6.0b1 (496 classes)
- struts-1.2.8 (273 classes)
- jcoverage-1.0.5 (90 classes)
- easymock-1.0.5 (47 classes)
- junit-3.8.1 (46 classes)

All these software products are written in an object-oriented language, since extracted results will be applied to previous works on this paradigm. In the study, we use Eclipse 3.1 and Metrics 1.3.6 plug-in as the metric collection tool. This issue limits examples to programs written in Java, although, in our opinion, the process is usable in other object-oriented languages³.

The selected metrics work on classes, choosing metrics related to size, complexity, cohesion, inheritance and specialization [2, 6]:

- NOF number of fields
- NOM number of methods
- WMC ciclomatic complexity
- LCOM lack of cohesion of methods
- DIT depth in the inheritance tree
- NSC number of children
- SIX specialization index
- NORM number of overridden methods

For each one of them, we obtain the values for several descriptive statistics: mean, bounded mean (removing 15% of the extreme values), standard deviation, lower quartile (Q1), median (Q2), upper quartile (Q3), minimum and maximum.

3.2 Partial Conclusions

From the previous results, see Fig. 1, we can say that:

- distributions are not symmetrical, differences between mean and median, and proximity of the median to Q3 quartile prove this. Most cases show distributions with positive asymmetry (distribution tail to the right side). These measures follow this kind of distribution as expected.
- differences between minimum and maximum values are large, and they are also very different in each product. This suggests dispersed data, with different thresholds.
- product size (number of classes) is slightly correlated with some metrics. Size could affect the thresholds. This is more noticeable in metrics such as NOF, NOM and WMC, with a high correlation among them. On the contrary, metrics such as LCOM and DIT show low variations between different products.

³ Number of classes is conditioned to the use of Metrics 1.3.6 plug-in, which does not count the number of inner classes

	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JFreeChart 1.0.0-pre2	2,40	10,08	22,98	0,21	2,55	0,36	0,16	0,69
Bounded mean (15%)	1,41	7,45	15,87	0,17	2,47	0,04	0,08	0,46
Q3	3,00	11,00	25,00	0,50	3,00	0,00	0,14	1,00
Median	1,00	5,00	9,00	0,00	3,00	0,00	0,00	0,00
Q1	0,00	3,00	6,00	0,00	2,00	0,00	0,00	0,00
Standard Deviation	5,05	15,01	38,82	0,32	1,14	1,48	0,37	1,23
Minimum	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
Maximum	48,00	166,00	490,00	1,00	7,00	16,00	3,20	9,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean Junit-3.8.1	2,17	8,13	15,70	0,21	2,70	0,28	0,18	0,35
Bounded mean (15%)	1,50	6,53	12,33	0,18	2,58	0,15	0,09	0,28
Q3	2,00	9,75	15,75	0,50	3,75	0,00	0,12	1,00
Median	1,00	4,50	8,00	0,00	2,00	0,00	0,00	0,00
Q1	0,00	2,00	4,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	3,59	10,35	20,42	0,33	1,84	0,72	0,45	0,60
Minimum	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	18,00	62,00	106,00	0,91	6,00	3,00	2,00	3,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean Jcoverage-1.0.5	1,49	4,35	9,56	0,24	1,78	0,39	0,81	0,28
Bounded mean (15%)	1,23	3,70	8,17	0,20	1,62	0,19	0,10	0,21
Q3	2,00	5,00	14,00	0,50	2,00	0,00	0,00	0,00
Median	1,00	3,00	5,00	0,00	1,00	0,00	0,00	0,00
Q1	0,00	2,00	3,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	1,87	4,46	9,59	0,34	1,05	0,96	0,37	0,52
Minimum	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	7,00	25,00	46,00	1,00	5,00	4,00	1,67	2,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean easymock-2.0	1,41	5,83	12,54	0,15	1,24	0,09	0,12	0,33
Bounded mean (15%)	1,24	4,13	8,32	0,11	1,08	0,00	0,02	0,16
Q3	2,00	5,00	13,50	0,33	1,00	0,00	0,00	0,00
Median	1,00	3,00	3,50	0,00	1,00	0,00	0,00	0,00
Q1	1,00	3,00	3,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	1,34	7,51	19,25	0,24	0,67	0,46	0,41	0,73
Minimum	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
Maximum	6,00	38,00	105,00	0,85	4,00	3,00	2,00	3,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean struts-1.2.8	2,91	8,60	18,84	0,28	2,59	0,46	0,51	0,96
Bounded mean (15%)	2,09	6,66	13,21	0,25	2,45	0,24	0,33	0,67
Q3	4,00	11,00	22,00	0,67	4,00	1,00	0,60	1,00
Median	2,00	4,00	8,00	0,00	2,00	0,00	0,00	0,00
Q1	0,00	2,00	3,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	4,56	11,02	29,13	0,36	1,48	1,13	0,95	2,04
Minimum	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
Maximum	40,00	82,00	260,00	0,98	7,00	10,00	5,00	28,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JHotDraw60b1	1,40	9,51	13,36	0,16	2,84	0,57	0,31	0,73
Bounded mean (15%)	1,09	7,72	10,31	0,11	2,68	0,07	0,16	0,38
Q3	2,00	11,00	14,00	0,00	4,00	0,00	0,32	1,00
Median	1,00	7,00	9,00	0,00	3,00	0,00	0,00	0,00
Q1	0,00	4,00	5,00	0,00	2,00	0,00	0,00	0,00
Standard Deviation	1,86	10,40	16,76	0,30	1,49	3,84	0,74	1,70
Minimum	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
Maximum	19,00	90,00	158,00	1,50	9,00	71,00	8,00	19,00

Fig. 1. Overall results

In previous works, we established the possibility to fix metric thresholds with the aim of detecting flaws (in design, not in functionality). A first approximation was to establish these thresholds on the basis of generic values. However, the results show that values should be fitted to the concrete product.

Another factor that could influence results is the kind of product. Similar products such as testing frameworks (junit & easymock), development frameworks (struts & jhotdraw) and libraries (jcoverage & jfreechart), present large differences between minimum and maximum values.

From these results, the hypothesis appears that the absence of thresholds may generate a large change of metric measures (while products increase their size, metrics could increase or decrease over the recommended values). To verify this hypothesis, we carry out a second case study with different versions of some products.

3.3 Second Phase: Version Evolutions

We take different versions of three products: JFreechart, JHotDraw and JUnit. We show the versions and number of classes of each version. These versions have evolved over a medium period of time: *jfreechart-0.9.4* (326 classes, 2002-10-18), *jfreechart-0.9.7* (492 classes, 2003-04-17), *jfreechart-0.9.21* (570 classes, 2004-09-10), *jfreechart-1.0.0-pre2* (629 classes, 2005-03-10) and *jfreechart-1.0.1* (691 classes, 2006-01-27).

	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jfreechart-0.9.4	2.69	7.77	18.00	0.26	3.02	0.40	0.27	0.61
Bounded mean (15%)	1.71	6.13	13.51	0.22	2.85	0.08	0.11	0.32
Q3	3.00	11.00	24.00	0.62	4.00	0.00	0.16	1.00
Median	1.00	4.00	8.00	0.00	3.00	0.00	0.00	0.00
Q1	0.00	1.00	3.00	0.00	1.00	0.00	0.00	0.00
Standard Deviation	4.87	9.70	25.11	0.35	1.95	1.49	0.70	1.34
Minimum	0.00	0.00	1.00	0.00	1.00	0.00	0.00	0.00
Maximum	39.00	60.00	195.00	1.00	7.00	16.00	6.00	8.00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jfreechart-0.9.7	2.14	7.03	15.63	0.20	3.21	0.31	0.17	0.49
Bounded mean (15%)	1.22	5.06	11.08	0.15	3.07	0.04	0.07	0.28
Q3	2.00	9.00	19.00	0.50	4.00	0.00	0.09	1.00
Median	0.00	3.00	6.00	0.00	3.00	0.00	0.00	0.00
Q1	0.00	1.00	3.00	0.00	2.00	0.00	0.00	0.00
Standard Deviation	4.55	10.65	24.45	0.32	1.97	1.34	0.44	1.02
Minimum	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Maximum	39.00	87.00	203.00	1.00	7.00	15.00	3.00	7.00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jfreechart-0.9.21	2.38	9.99	22.47	0.21	2.52	0.36	0.16	0.66
Bounded mean (15%)	1.41	7.33	15.44	0.17	2.45	0.05	0.08	0.44
Q3	2.00	12.75	26.00	0.50	3.00	0.00	0.16	1.00
Median	1.00	5.00	9.00	0.00	3.00	0.00	0.00	0.00
Q1	0.00	3.00	6.00	0.00	2.00	0.00	0.00	0.00
Standard Deviation	4.93	15.20	38.66	0.32	1.12	1.47	0.37	1.20
Minimum	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Maximum	47.00	155.00	473.00	0.96	7.00	16.00	3.00	8.00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JFreeChart-1.0.0-pre2	2.40	10.08	22.98	0.21	2.55	0.36	0.16	0.69
Bounded mean (15%)	1.41	7.45	15.87	0.17	2.47	0.04	0.08	0.46
Q3	3.00	11.00	25.00	0.50	3.00	0.00	0.14	1.00
Median	1.00	5.00	9.00	0.00	3.00	0.00	0.00	0.00
Q1	0.00	3.00	6.00	0.00	2.00	0.00	0.00	0.00
Standard Deviation	5.05	15.01	38.82	0.32	1.14	1.48	0.37	1.23
Minimum	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Maximum	48.00	166.00	490.00	1.00	7.00	16.00	3.20	9.00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jfreechart-1.0.1	2.22	9.94	22.42	0.19	2.53	0.33	0.16	0.69
Bounded mean (15%)	1.27	7.27	15.25	0.15	2.46	0.03	0.09	0.48
Q3	2.00	11.00	23.00	0.40	3.00	0.00	0.17	1.00
Median	1.00	5.00	9.00	0.00	3.00	0.00	0.00	0.00
Q1	0.00	4.00	7.00	0.00	2.00	0.00	0.00	0.00
Standard Deviation	4.86	15.18	39.39	0.31	1.12	1.41	0.35	1.18
Minimum	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Maximum	46.00	173.00	513.00	1.00	7.00	14.00	3.33	8.00

Fig. 2. JFreeChart evolution results

In the case of JHotDraw, version, number of classes and dates are: *jhotdraw-5.2* (149 classes, 2001-02-18), *jhotdraw-5.3* (208 classes, 2002-01-20), *jhotdraw-5.4b1* (478 classes, 2004-01-31) and *jhotdraw-6.0b1* (497 classes, 2004-02-01).

In the case of JUnit, versions and number of classes are⁴: *junit-2.1* (19 classes), *junit-3.8.1* (47 classes) and *junit-3.2* (32 classes).

For each of these products, we collected previously mentioned metrics, obtaining mean, bounded mean, standard deviation, Q1, median (Q2), Q3, minimum

⁴ Product release dates are not available

	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jhotdraw52	1,83	8,30	13,53	0,26	2,81	0,68	0,56	1,28
Bounded mean (15%)	1,52	6,37	10,25	0,23	2,60	0,24	0,48	1,06
Q3	3,00	10,00	15,25	0,60	3,00	0,00	1,00	2,00
Median	1,00	5,00	8,00	0,00	2,00	0,00	0,28	1,00
Q1	0,00	3,00	4,00	0,00	2,00	0,00	0,00	0,00
Standard Deviation	2,19	9,82	16,84	0,33	1,70	1,91	0,66	1,69
Minimum	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	14,00	61,00	108,00	1,50	8,00	12,00	3,11	12,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jhotdraw53	1,83	9,12	15,51	0,27	2,65	0,86	0,51	1,21
Bounded mean (15%)	1,46	7,07	11,60	0,23	2,43	0,20	0,41	0,97
Q3	3,00	10,00	18,00	0,63	3,00	0,00	0,75	2,00
Median	1,50	6,50	12,50	0,00	1,00	0,00	0,00	0,00
Q1	0,00	3,00	4,75	0,00	2,00	0,00	0,00	0,00
Standard Deviation	2,38	10,87	20,54	0,35	1,66	3,53	0,66	1,66
Minimum	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	17,00	72,00	146,00	1,50	8,00	40,00	3,00	12,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean jhotdraw54b1	1,41	9,67	13,89	0,16	2,90	0,58	0,32	0,73
Bounded mean (15%)	1,12	7,85	10,80	0,11	2,75	0,08	0,17	0,39
Q3	2,00	11,00	15,00	0,00	4,00	0,00	0,33	1,00
Median	1,00	7,00	9,00	0,00	3,00	0,00	0,00	0,00
Q1	1,00	4,00	6,00	0,00	2,00	0,00	0,00	0,00
Standard Deviation	1,81	10,33	16,88	0,30	1,48	3,89	0,75	1,71
Minimum	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
Maximum	16,00	88,00	148,00	1,50	9,00	71,00	8,00	19,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JHotDraw60b1	1,40	9,51	13,36	0,16	2,84	0,57	0,31	0,73
Bounded mean (15%)	1,09	7,72	10,31	0,11	2,68	0,07	0,16	0,38
Q3	2,00	11,00	14,00	0,00	4,00	0,00	0,32	1,00
Median	1,00	7,00	9,00	0,00	3,00	0,00	0,00	0,00
Q1	0,00	4,00	5,00	0,00	2,00	0,00	0,00	0,00
Standard Deviation	1,86	10,40	16,76	0,30	1,49	3,84	0,74	1,70
Minimum	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
Maximum	19,00	90,00	158,00	1,50	9,00	71,00	8,00	19,00

Fig. 3. JHotDraw evolution results

and maximum, as can be seen in Fig. (2, 3, 4). In Fig. 5, the JFreeChart evolution example (using mean value) shows that the five versions have similar values over four years. This suggests that stable products maintain their thresholds, even after increasing their size (duplicating the size in all cases).

3.4 Conclusions of the Study

These are the conclusions extracted from the study on several versions:

- Thresholds should be relative to the product.
- Thresholds could be maintained between different stable versions.
- The kind of product (framework / library) does not determine how we should fix its thresholds.

From these conclusions, we settle new issues. To develop new products we need to tune new initial thresholds. As a first option, we can estimate values from similar products (same domain, similar functionality and similar size) by taking into account our previous results. If several product versions are available, we can collect values from them to calculate an initial estimation. In both cases, we probably need to fix the values.

	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JUnit 2.1	2,16	8,11	14,05	0,22	2,53	0,32	0,31	0,58
Bounded mean (15%)	1,35	7,18	12,41	0,19	2,41	0,18	0,17	0,47
Q3	2,00	8,50	18,00	0,50	3,00	0,00	0,26	1,00
Median	1,00	4,00	6,00	0,00	2,00	0,00	0,00	0,00
Q1	0,00	4,00	4,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	4,06	8,52	15,30	0,31	1,61	0,82	0,70	0,84
Minimum	0,00	1,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	18,00	31,00	55,00	0,89	6,00	3,00	3,00	3,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JUnit 3.2	2,72	7,94	14,75	0,25	2,56	0,19	0,13	0,34
Bounded mean (15%)	1,68	5,96	11,29	0,22	2,43	0,04	0,09	0,25
Q3	3,00	11,00	18,50	0,50	3,50	0,00	0,19	1,00
Median	1,00	3,50	5,50	0,00	2,00	0,00	0,00	0,00
Q1	0,00	2,00	2,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	4,85	11,65	21,05	0,34	1,93	0,64	0,24	0,65
Minimum	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	20,00	60,00	103,00	0,92	6,00	3,00	0,75	3,00
	NOF	NOM	WMC	LCOM	DIT	NSC	SIX	NORM
Mean JUnit 3.8.1	2,17	8,13	15,70	0,21	2,70	0,28	0,18	0,35
Bounded mean (15%)	1,50	6,53	12,33	0,18	2,58	0,15	0,09	0,28
Q3	2,00	9,75	15,75	0,50	3,75	0,00	0,12	1,00
Median	1,00	4,50	8,00	0,00	2,00	0,00	0,00	0,00
Q1	0,00	2,00	4,00	0,00	1,00	0,00	0,00	0,00
Standard Deviation	3,59	10,35	20,42	0,33	1,84	0,72	0,45	0,60
Minimum	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,00
Maximum	18,00	62,00	106,00	0,91	6,00	3,00	2,00	3,00

Fig. 4. JUnit evolution results

4 Applying Relative Thresholds

In previous works, we tackled the usefulness of using metrics as symptoms of bad smells. This term is restricted to refactoring, although it could be generalized to software flaws. We posed the use of thresholds to suggest them. We defined a framework which supports all these concepts. Nevertheless, threshold definition is an open question [5]. In Fig. 6, we have the box plot diagrams of two data distributions: ideal distribution (positive gamma distribution without outliers) and actual distribution of WMC metric in JFreeChart 1.0.1. We consider as low and high values those below and above Q1 and Q3 respectively. More concretely, in these subsets, the outliers are the main candidates to point to problematic components. In an ideal process, outliers should be removed by applying detection and correction of bad smells.

We formerly concluded that the application of the same values to different products does not seem to be adequate. Tuning values should be helped. Next, we show a review of previous works, applying the results obtained so far.

4.1 Detection of “Lazy Classes”

Reviewing again the definition [4], there are classes that *“are not doing enough to pay by themselves should be eliminated”*. The established criteria are a low number of methods and fields, low complexity and a high level (low value) in

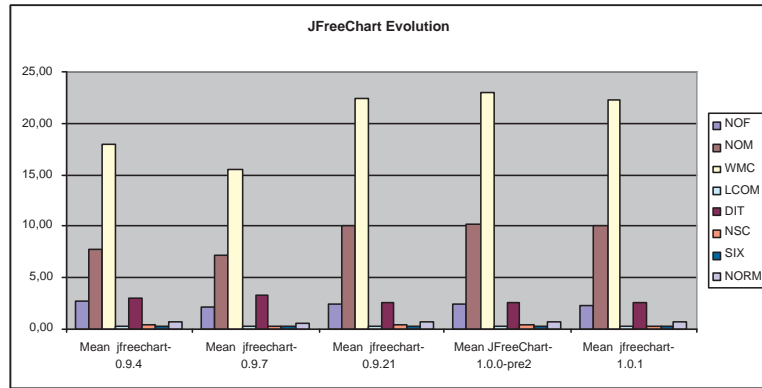


Fig. 5. JFreeChart evolution

the inheritance hierarchy. Furthermore, an additional criterion could be added, such as a low number of children.

In this approach, the problem is: What is considered low in this system? As pointed out in previous works [8, 10], certain systems are particular and general conclusions may not be correct. In our case, we work with quartiles to associate low values with classes below the Q1 quartile. We have the three Q1 quartiles of NOF, NOM and WMC as limits.

As can be seen, there are certain disagreements in the filters to be used ($\text{NOF} \leq Q1$ AND $\text{NOM} \leq Q1$ AND $\text{WMC} \leq Q1$). For instance, what happens if we apply the JUnit collected values to JFreeChart? In this case, we mark as suspect 63 classes, whereas, using its own values, 97 classes are selected. The difference in numbers explains why we should not apply the same criteria to the two products.

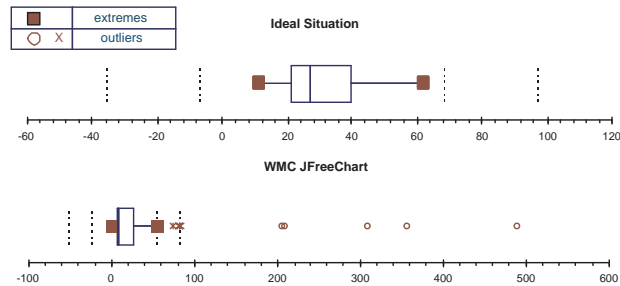


Fig. 6. Box Plot diagrams

4.2 Detection of “Large Classes”

At first, it seemed to be the easiest bad smell to detect. However, as we saw above, each system has its own particularities. More concretely, if we establish the same threshold for all of them, it is possible to fix a high value that does not return any result. On the other hand, if we choose a low value, too many classes are selected in other sets.

Using the knowledge about data distribution, we look for extreme values. We work with quartiles to associate high values with classes above the Q3 quartile. We establish the Q3 quartile as the threshold value for each product, combining NOF/NOM/WMC metrics.

From the study of the values, we infer the problem of applying the filter values to JFreeChart vs. jcoverage, or vice versa. Those metric values in the righthand tail of the distribution will verify the filter ($\text{NOF} \geq \text{Q3}$ AND $\text{NOM} \geq \text{Q3}$ AND $\text{WMC} \geq \text{Q3}$). It should be possible to fit values to find outliers. If we repeat the process, for example the JUnit filter to JFreeChart, we find 148 suspect classes. However, applying the filter to JFreeChart we have just 108 classes.

Results show large differences between applying one or other threshold. The final accuracy, however, depends on manual tuning, taking into account the number of false positives and true negatives.

5 Conclusions and Future Works

Current work fixes, from a case study, the suitability of using generic vs. relative product thresholds. The former solution, generic thresholds, has not been completely abandoned. We continue to give support with metric profiles, although each product usually has its own limits.

Highly different results among software products lead us to assume it is not completely correct to use them as discriminants in the detection of bad smells. Product size, in many cases, limits the values of the metrics. Nevertheless, other metrics seem to be less sensitive to these effects.

In this work, we do not pretend to obtain new thresholds, or new methods to define them. We want to check out, empirically, the suitability of their definition for each kind of product. This detection process should be repeated until stable distributions are achieved, so as to reduce the number of outliers. These results could also help in the systematic software maintenance, as long as we have previous stable versions. Obviously, further analysis on larger samples should be completed to confirm these results.

Finally, we propose to include the current solution in bad smell detection, alongside our metric collection framework. Final tools should be able to aid the users to establish their own criteria in an objective way.

Obviously, there are many lines of work still open:

- We need to validate results, increasing the number of products under study.
- We should check the language influence on the results.
- Experience, knowledge and culture of the programmer could influence the software evolution.

References

1. Francisca Muñoz Bravo. *A Logic Meta-Programming Framework for Supporting the Refactoring Process*. PhD thesis, Vrije Universiteit Brussel, Belgium, 2003.
2. Shyam R. Chimdaber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions On Software Engineering*, 20:476–493, 1994.
3. Yania Crespo, Carlos López, Raul Marticorena, and Esperanza Manso. Language independent metrics support towards refactoring inference. In *9th ECOOP Workshop on QAOOSE 05 (Quantitative Approaches in Object-Oriented Software Engineering)*. Glasgow, UK. ISBN: 2-89522-065-4, pages 18–29, jul 2005.
4. Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Number 0-201-48567-2. Addison-Wesley, 2000.
5. V.A. French. Establishing software metric thresholds. *9th International Workshop on Software Measurement*, 1999.
6. Mark Lorenz and Jeff Kidd. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.
7. Mika Mäntylä. *Bad Smells in Software - a Taxonomy and an Empirical Study*. PhD thesis, Helsinki University of Technology, 2003.
8. Mika Mäntylä, Jari Vanhanen, and Casper Lassenius. Bad smells - humans as code critics. In *20th IEEE International Conference on Software Maintenance*, 2004.
9. Radu Marinescu. Detecting design flaws via metrics in object-oriented systems. In *Proceedings of the TOOLS*, USA 39, Santa Barbara, USA, 2001.
10. Radu Marinescu. *Measurement and Quality in Object-Oriented Design*. PhD thesis, Faculty of Automatics and Computer Science, october, 2002.
11. Raul Marticorena. Analysis and definition of a language independent refactoring catalog. In *17th Conference on Advanced Information Systems Engineering (CAiSE 05)*. Doctoral Consortium, Porto, Portugal., page 8, jun 2005. <http://gnomo.fe.up.pt/caise/>.
12. Tom Tourwé and Tom Mens. Identifying Refactoring Opportunities Using Logic Meta Programming. In *Proc. 7th European Conf. on Software Maintenance and Reengineering*, pages 91 – 100, Benvento, Italy, 2003. IEEE Computer Society.