2011 Fourth International Conference on Software Testing, Verification and Validation Workshops

An experience report on using code smells detection tools

Francesca Arcelli Fontana, Elia Mariani
Università of Milano Bicocca
Department of Computer Science
Milano, Italy
arcelli@disco.unimib.it

Andrea Morniroli, Raul Sormani, Alberto Tonello
University of Milano Bicocca
Department of Computer Science
Milano, Italy
a.morniroli@campus.unimib.it

***Abstract*—Detecting code smells in the code and consequently applying the right refactoring steps when necessary is very important to improve the quality of the code. Different tools have been proposed for code smell detection, each one characterized by particular features. The aim of this paper is to describe our experience on using different tools for code smell detection. We outline the main differences among them and the different results we obtained.**

Keywords- code smells; code smell detection toools; refactoring; quality code evaluation.

I. INTRODUCTION

Code smells are characteristics of software that may indicate a code or design problem and can make software hard to evolve and maintain. Many smells have been defined in [4], others have been identified in the literature later as for example in [11] and new ones can be discovered.

To eliminate code smells and hence improve maintainability and software evolution we have to apply refactoring steps to enhance the inner quality of software. Many papers in the literature have been proposed on these topics as for example [5,11,12,15]. A smell is a symptom, refactoring is a cure. The same symptom may have a different cure**,** thus every smell may suggest more than one refactoring.

Several considerations have to be made:

- not necessarily all the code smells have to be removed, it depends on the system, sometimes the smell cannot be removed, it is the best solution; a typical example is given, in certain cases, by the code smell Large Class (see [4,12] for a list with the descriptions of the smells we consider in this paper). For example, a Large Class could be acceptable when it is used to implement a parser.
- Detecting smells is hard and costly. Hence when they have to be removed, it is better to remove them as early as possible. Tool support for their detection is particularly useful, since many code smells can go unnoticed while programmers are working.
- Code smells are not formally defined, hence their detection can provide uncertain and unsafe results (as we will see in this paper by experimenting with detection tools). Many problems can occur in interpreting the definitions of the smells [14]: there are some words and constraints in the definitions of smells such as the words *few*, *enough*, *large* which are obviously ambiguous and can be evaluated, for example, through fuzzy logic.
- The different detection techniques used by the detection tools are usually based on the computation of a particular set of combined metrics, or standard object-oriented metrics, or metrics defined ad hoc for the smell detection purpose [7]. This represents a huge problem since we have to define the threshold values for these metrics, which is particularly relevant for some smells. To establish these values different factors have to be taken into account, such as the system domain and size, organization best practices, the expertise and the understanding of the software engineers and the programmers who define these values (for example the threshold for Long Method or Large Class smell: when we have to consider a long method or a large class). Changing these thresholds obviously has a great impact on the number of detected smells (too many smells or lost smells).
- Usually other information for code smell detection, other than metric combination, is not taken into account, as for example relations existing between classes and other smells.
- Since several tools have been developed, one would like to use the "best one": which is the best one? For this aspect we have to face the common and very difficult problem of tool comparison. We faced the same problem for design pattern detection tools because a widely accepted benchmark platform for design patterns detection is not yet available, but some approaches have been proposed, as for example [1]. For code smell detection tools a benchmark

978-0-7695-4345-1/11 $26.00 © 2011 IEEE
DOI 10.1109/ICSTW.2011.12

platform has not been developed yet, the comparison is a difficult task for many reasons as those cited above. In general the validation of the results of the existing approaches is scarce, done only on small systems and for few smells. Other than the above cited problems on ambiguity on smell definitions and fixing the thresholds for the metrics, the value of some metrics, like those used to measure method complexity, can change from a manual inspection done by a programmer or the automatic computation of this metric done by a tool. Moreover, the manual computation can differ from one programmer to another, it is subjective. A first validation with precision and recall results is described in [14]. Usually the recall is higher for the smells detected through simple metrics (i.e. counting entities).

- Some detection tools, as we will see, have been developed to improve code quality during software development, other tools to support reengineering and maintenance activities.
- Most of the tools are not able to perform refactoring automatically on detected smells (for example JDeodorant is one of the tools described below which provides refactoring choices), but modern IDEs are usually capable of performing refactoring automatically, also if they need user guidance. It would be desirable for smell detector tools to help understand at least the cause of the smells and that the smell detector should not display smell information in such a way that a proliferation of code smells overloads the programmer, as underlined in useful guidelines proposed in [16].

The principal aims of this paper are the following:

- briefly introduce the principal code smell detection tools available, certainly the list is not complete, but we hope to have at least cited the most important ones (Section II);
- describe the experiments we did with some of these tools on several systems, also if in this paper we focus our attention on different versions of GanttProject system (see Section III-A, III-B);
- according to the experiments and results obtained, we conclude by remarking the reasons of the differences across the smell detection tools, the advantages or disadvantages of them and *what is missing*. (Section IV).

II. CODE SMELL DETECTION TOOLS

Many tools for code smell detection have been provided. We briefly describe below the tools that we experimented with and then we cite the other main tools available. We concentrate our experimentation on tools for code smell detection for Java systems. In Table 1 we describe the main features of these tools and the smells they detect.

JDeodorant [21] is an eclipse plug that automatically identifies four code smells (see Table 2) in Java programs, it ranks the refactoring that resolves the identified problems according to their impact on the design and automatically applies the most effective refactoring. Moreover, the tool assists the user in determining an appropriate sequence of refactoring applications.

PMD [19**]** scans Java source code and looks for potential problems like: possible bugs, such as dead code, empty try/catch/finally/switch statements, unused local variables, parameters and duplicate code. Moreover, PMD is able to detect three smells (see Table 1) and allows to set the thresholds values for the metrics.

iPlasma [10, 17] is an integrated platform for quality assessment of object-oriented systems that includes support for all the necessary phases of analysis: from model extraction up to high-level metrics based analysis, or detection of code duplication. iPlasma is able to detect several code smells called disharmonies, which are classified in identity disharmonies, collaboration disharmonies and classification disharmonies. The detailed description of these disharmonies can be found in [7].

InFusion [9] supports the analysis, diagnosis quality improvement of a system at the architectural, as well as at the code level and covers all the necessary phases of the analysis process. InFusion allows to detect more than 20 design flaws and code smells, like Code Duplication, classes that break encapsulation (Data Class, God Class), methods and classes that are heavily coupled or ill-designed class hierarchies and other code smells(see Table 1). InFusion has its roots in iPlasma, then extended with more functionalities.

StenchBlossom [16**]** is a smell detector that provides an interactive ambient visualization designed to first give programmers a quick, high-level overview of the smells in their code, and then, to help understand the sources of the code smells, but is not suitable for reverse engineering. It does not provide numerical values, but only a visual threshold: the size of a petal is directly proportional to the entity of the code smells. The only possible procedure to find code smells is to manually browse the source code, looking for a petal whose size is big enough to make us suppose that there is a code smell. It is a plugin for the Eclipse environment that provides the programmer with three different views, which progressively offer more information about the smells in the code being visualized. The tool is able to detect 8 smells (see Table 2).

Other developed tools or methods for code smell detection have been proposed, as for example: DECOR-Black-Box [13,14], an approach that allows the

specification and automatic detection of code and design smells (also called anti patterns). In particular they specified six code smells, automatically generated their detection algorithms using templates, and validated the algorithms in terms of precision and recall. In the following, with the name DECOR we mean the component developed for smell detection. CodeVizard [18] is a tool which can detect several smells essentially following the detection rules defined in [7]. We would like to experiment with this tool, but it is not available yet. CheckStyle [3] has been developed to help programmers write Java code that adheres to a coding standard. It is able to detect duplicate code and three other smells, Long Method, Long Parameter List and Large Class. InCode [8] is an Eclipse plugin that provides continuous detection of design problems (i.e. problems are detected as code is written) complementing thus the code reviews, which can be performed with other tools. It has been developed by the same team of inFusion and is very similar to Infusion for the functionalities provided for our tasks. Then other tools performing related tasks are: FxCop for .Net, Analyst for Java a commercial tool, CodeNose, no longer available, JCosmo for Linux, CloneDigger and ConQat for clone code detection.

III. EXPERIMENTING CODE SMELLS DETECTION TOOLS

In the following we report the results of the experiments we did on different versions of GanttProject, a multi-application platform for planning and project management, by exploiting iPlasma, InFusion, Jdeodorant, Stench Blossom, and PMD.

A. Experiments set up

We will observe how the tools report different results, also by analyzing the same system. We also did the same experiment on different versions of other systems, but in this paper we focus our attention on GanttProject, as an object-oriented open source system. Moreover for a version of this system, GanttProject v1.10.2, in [14] the authors provided a validation and a comparison between DECOR and iPlasma (in the following we call this validation DECOR benchmark). The authors claim that this validation constitutes a first result in the literature on both the precision and recall of a detection technique on open-source systems and on a representative set of six smells. This is the reason why we decided to analyze GanttProject. For at least one version and at least six smells we know the number of smells detected in the system.

The smells detected by each one of the tools we have considered in our experimentation are shown in Table 2 (in this table we list the smells detected at least by two tools). The versions of GanttProject which we analyzed are those specified in Table 3. Also, whenever possible, the DECOR's benchmark performed on GanttProject 1.10.2 is used as a reference when analyzing the results of the tests.

Table 1. Code smells detection tools

Tool	Type	Version	Supported Languages	Code smells detected	Automated refactoring	Direct link to code
JDeodorant	Eclipse Plug-in (vv. 3.5 e 3.6)	4.0.4 2010	Java	Feature Envy God Class Long Method Type Checking	Yes	Yes
Stench Blossom	Eclipse Plug-in (v. 3.3)	1.0.4 2009	Java	Data Clumps Feature Envy InstanceOf Long Method Large Class Message Chain Switch Statement Typecast	No	Yes
InFusion	Stand-alone application	7.2.11 2010	C, C++, Java	Cyclic Dependencies Brain Method Data Class Feature Envy God Class Intensive Coupling Missing Template Method Refused Parent Bequest Significant Duplication Shotgun Surgery	No	No
iPlasma	Stand-alone application	6.1 2009	C++, Java	Default: Brain Class Brain Method Data Class Dispersed Coupling Feature Envy God Class Intensive Coupling Shotgun Surgery Refused Parent Bequest Tradition Breaker By custom rules: Long Method Long Parameter List Speculative Generality	No	No
PMD	Eclipse Plugin or Stand-alone application	4.2.5 2009	Java	Large Class Long Method Long Parameter List	No	Yes
DÉCOR (Black Box)	Stand-alone application.	1.0 2009	Java	Large Class Lazy Class Long Method Long Parameter List Refused Parent Bequest Speculative Generality	No	No

As reported in Table 2, there is no code smell shared by all the tools. Due to this reason, the following analysis is performed on subsets of the tools. The experiments were done on PC RAM 2GB, IntelCore 2,T7200, BUS 667.

It is important to observe that, sometimes code smells are called in a different way by the tools even if they share the same features.

Table 2: Code smells detected by the tools	
Code smell	**Used tools**
Brain Method	InFusion, iPlasma
Data Class	InFusion, iPlasma
Feature Envy	JDeodorant, Stench Blossom, InFusion, iPlasma
God Class / Large Class (DECOR)	JDeodorant, InFusion, iPlasma, DECOR (Benchmark)
Intensive Coupling	InFusion, iPlasma
Large Class	Stench Blossom, PMD
Long Method	JDeodorant, Stench Blossom, iPlasma, PMD, DECOR (Benchmark)
Long Parameter List	iPlasma, PMD, DECOR (Benchmark)
Refused Parent Bequest	InFusion, iPlasma, DECOR (Benchmark)
Shotgun Surgery	InFusion, iPlasma
Speculative Generality	iPlasma, DECOR (Benchmark)

Table 3: Dimensions of the versions of GanttProject					
Metric	**GanttProject version**				
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Packages	27	27	27	28	33
Classes	393	395	396	402	549
Methods	1913	1921	1924	1937	2724

Moreover, code smells with different features could share the same name. For instance:

- The Code smell Large Class detected by DECOR is equivalent to the God Class smell recognized by JDeodorant, inFusion and iPlasma. In fact, both specifications define classes that are trying to do too much. These classes also have low cohesion.
- The Code smell Large Class detected by Stench Blossom and PMD is different from Large Class recognized by DECOR. Indeed, Stench Blossom and PMD simply concern Large Class as a class with many lines of code, whereas DÉCOR consider both the size in terms of number of methods and attributes and the cohesion of the class.

There are also remarkable differences concerning the number of classes and methods reported by each tool on the releases of GanttProject. Since documentation regarding this matter is usually missing, we decided to test the code smells reported in Table 2 and to use another tool, *Understand* (SciTools), as a reference for the computation of the number of packages, classes and methods.

B. Experiments results.

For each smell in Table 2 we experimented the tools that can detect the smell; for each one we show a Figure and one or more Tables (see the end of the paper), which outline the differences in the detected smells in the 5 versions of GanttProject.

Brain Method (Figure 1, Table 6): InFusion and iPlasma detect the same number of this smell for each version of GanttProject due to the fact that they both use the same metrics (for this reason we show in Figure 1 only one line). We cannot compare the results with DECOR benchmark on Release 1.10.2 because DECOR does not consider Brain Methods.

Data Class (Figure 2, Table 7): InFusion and iPlasma detect a very similar number of Data Classes in each version of GanttProject, since both use the same metrics. As we can see from the table, the different results regard only three possible smells (the differences in the total number of smells is always three). We analyzed the code, and this difference always regards the following same classes:AboutFieldTableModel,AboutLibraryPanel.LibraryInfo, AuthorsFieldTableModel.

Feature Envy (Figure 3, Table 8): Surprisingly, the values identified by InFusion and iPlasma are very different. We suppose that even if they use the same metrics, they use different threshold values. JDeodorant, for some GanttProject releases detects the same values as InFusion; for other versions we have very different values. Stench Blossom detects more Feature Envy code smells than other tools, but we have to remark, as outlined in Section 2, that the computation of the number of smells is done manually, browsing the source code and looking for petals whose size is big enough to make us suppose that there is a code smell. Thus, we could have made mistakes, because it is too subjective.

God Class/ Large Class (DECOR) Figure 4, Table 9, Table 10: The number of God Classes in the different releases of GanttProject remains constant. InFusion and iPlasma results are very close to each other and similar to those of DECOR benchmark. JDeodorant has remarkable differences compared to the other tools.

Intensive Coupling (Figure 5, Table 11): as we can see from the table, the amount of Intensive Couplings detected by InFusion and iPlasma is the same.

Large Class (Figure 6, Table 12): in PMD, the threshold value (number of lines of code) has been set to 415. It is difficult to compare the results of PMD with those of Stench Blossom even if they are very similar, since we have to make the same comment as previously for the Feature Envy.

Long Method (Figure 7, Table 13, Table 14): Given the possibility of PMD to manually set the detection threshold (number of lines of code), various tests were carried out with different thresholds. The threshold used to analyze the 5 versions of GanttProject is 48, as it provides results similar to those found in the DECOR benchmark on GanttProject 1.10.2. Tests were completed manually, counting the Long Methods for each class. The tool's interface is not user-friendly and changing the thresholds requires the recompilation of the source code. As far as iPlasma is concerned, Long Method is not among the

predefined Code smells, and to detect it, we must manually set the filter. The differences between the values reported by the tools are due to the different detection techniques. Specifically, PMD and Stench Blossom simply count the number of lines of code of the methods (excluding comments), while iPlasma uses more complex metrics, described in detail in [7].

Long Parameter List (Figure 8, Table 15, Table 16): Both iPlasma and PMD allow to set the detection threshold of the Long Parameter List (number of parameters). We chose the value of 4 for the threshold, which was also used in the DECOR benchmark. In addition, in iPlasma to identify the Long Parameter List, we must manually set a detection rule, since it is not one of the default code smells (see Table 1). Although we used the same threshold, the results of PMD differ from those of the DECOR benchmark and iPlasma.

Refuse Parent Bequest (Figure 9, Table 17, Table 18): DECOR detects 13 code smells, but it is strange that iPlasma and InFusion detect, respectively, 1 and 0 smells. Since we do not have access to the specification of the code smell, we cannot understand the reasons for this big difference in the results.

Shotgun Surgery (Figure 10, Table 19): The values obtained by InFusion and iPlasma are very different. As in the case of Feature Envy, it is highly probable that even if they use the same metrics, they use different thresholds.

Speculative Generality (Figure 11, Table 20): Speculative Generality is not among the default iPlasma's code smells and the rule to locate them has to be specified. The number of these smells through the DECOR benchmark is four and through iPlasma nine.

IV. CONCLUDING REMARKS AND FUTURE DEVELOPMENTS

Concluding, we have to assert that comparing the tools is very difficult, and in some cases also using them is not very easy and immediate. Often the documentation is not adequate or is absent. The aim of this paper was not to compare the tools, but to describe our experience in using them and outline the difficulties in the comparison task. We think that the main critical aspect is to have no immediate visibility on the deduction rules used and on the thresholds of the metrics. It would be nice to have at least the possibility to set and change the thresholds according to the contexts and to have an automatically generated results report and, very important, the link to the code.

An analysis of the inconsistency between the tools in identifying the number of packages, classes and methods is significant and obviously may have distorted the percentage of smells calculated. Probably the tools are treating differently anonymous inner hidden classes which are typically found in GUI applications. For this reason we are interested in performing the same experimentation by studying a second non GUI application, such as ANT.

We would like to refine these experiments and going into the details of the different deduction rules and metric thresholds used. We also plan to consider other detection tools, such as those cited at the end of Section 2. Moreover to study possible improvements of the detection techniques, we would like to analyze the relations between code smells, between code smells and micro patterns [6] and the relations between code smells and anti patterns,also called design smells by many authors, since several antipatterns can be detected through the detection of code smells. These researches will be done within our Marple project activities [2].

REFERENCES

[1] F. Arcelli, M. Zanoni, A.Caracciolo, A benchmark platform for design patterns detection, *Procceedings of the PATTERNS 2010 Conference,* Lisboa, Portugal November 2010.

[2] F. Arcelli, M. Zanoni, A Tool for Design Pattern Detection and Software Architecture Reconstruction. *Information Sciences* Elsevier, doi:10.1016/j.ins.2010.

[3] CS-CheckStyle: http://checkstyle.sourceforge.net/index.html.

[4] Martin Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley , Inc., Boston, MA, USA, 1999.

[5] Hamza, H., Counsell, S., Hall, T. and Loizou, G., Code smell eradication and associated refactoring, 2nd European Computing Conference, 2008.

[6] Y.Gil, I. Maman, Micro Patterns in Java Code, *Proc. of the 20th annual ACM SIGPLAN conference* OOPSLA 2005, October 2005.

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

[8] InCode : http://www.intooitus.com/inCode.html

[9] InFusion: http://www.intooitus.com/inFusion.html

[10] iPlasma : http://loose.upt.ro/iplasma/index.html

[11] Mika V. Mäntylä, Jari Vanhanen, and Casper Lassenius. Bad smells – humans as code critics. *Proc. IEEE International Conference on Software Maintenance*, 0:399–408, 2004.

[12] T. Mens, T. Tourwé, A Survey of Software Refactoring, *IEEE Transactions on Software Engineering*, vol. 30, no. 2, 2004.

[13] Naouel Moha,Yann-Gael Gueheneuc, Laurence Duchien, Anne-Francoise Le Meur, Decor : a method for the specificatio and detection of code and design smells, *TSE, IEEE* CS Press, 2010.

[14] Naouel Moha, Yann-Gael Gu´eh´eneuc, Anne-Francoise Le Meur,Laurence Duchien and Alban Tiberghien, From a domain analysis to the specification and detection of code and design smells, *Formal Aspects of Computng*, 2009.

[15] Emerson Murphy-Hill, Chris Parnin, and Andrew P. Blac How we refactor, and how we know it. In ICSE '09 *Proceedings of the 31st International Conference on Software Engineering*, 2009.

[16] Emerson Murphy-Hill and Andrew P. Black, An interactive ambient visualization for code smells, Proceedings of SOFTVIS '10, USA, October 2010.

[17] C. Marinescu, R. Marinescu, P. Mihancea, D. Ratiu, and R. Wettel. iplasma: An integrated platform for quality assessment of object-oriented design. In *Proceedings of 21st International Conference on Software Maintenance (ICSM 2005), Tools Section*, 2005.

[18] Steffen Olbrich, Daniela Cruzes, Victor Basili, Nico Zazworka, CodeVizard:A tool to aid the analysis of software evolution, *Proceedings ICSE 2010*, South Africa.

[19] PMD http://pmd.sourceforge.net/

[20] http://essere.disco.unimib.it/reverse/CodeSmells .html

[21] Nikolaos Tsantalis, Theodoros Chaikalis, and Alexander Chatzigeorgiou. JDeodorant: Identification and removal of type-checking bad smells. In *Proceedings of CSMR* 2008, pp 329–331.

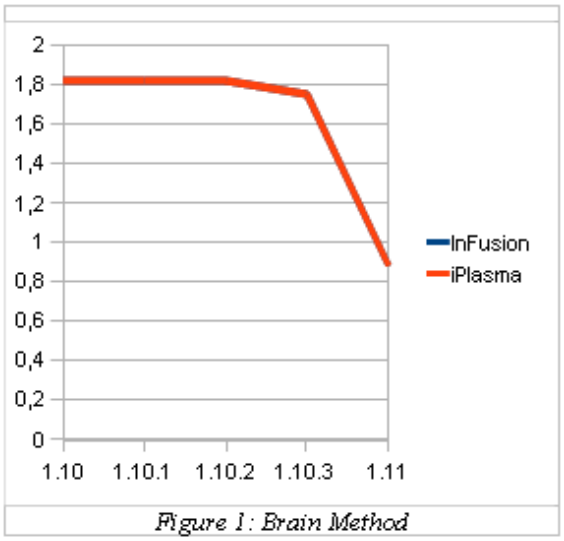

Figure 1: Brain Method

Figure 4: God Class

Figure 2: Data Class

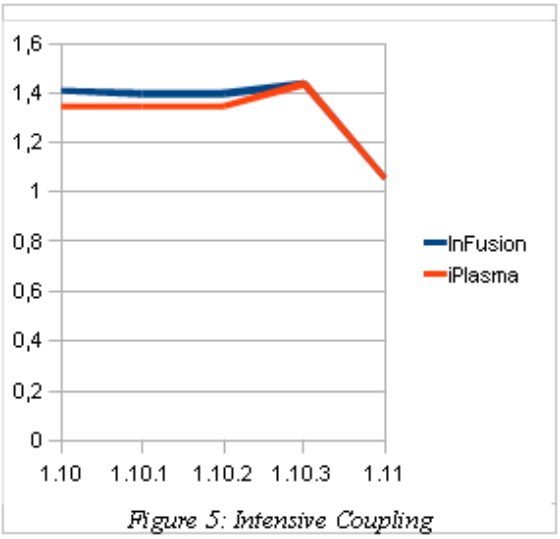

Figure 5: Intensive Coupling

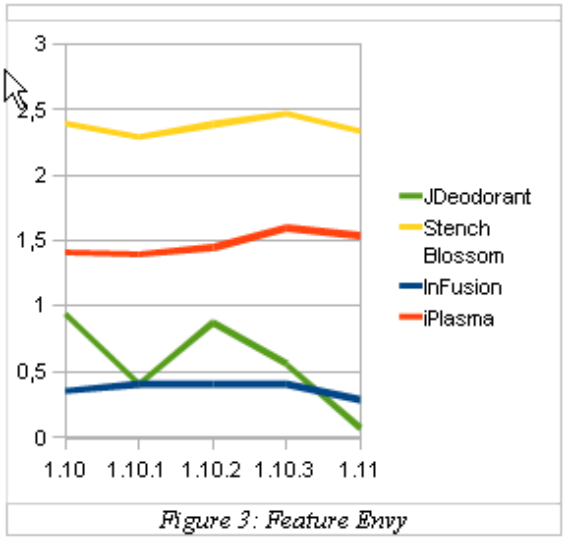

Figure 3: Feature Envy

Figure 6: Large Class

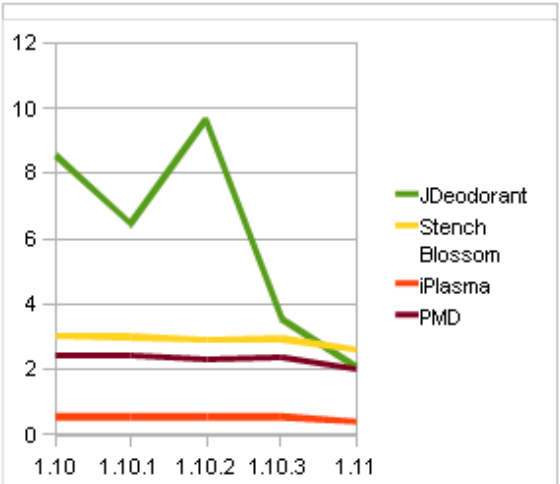

Figure 7: Long Method

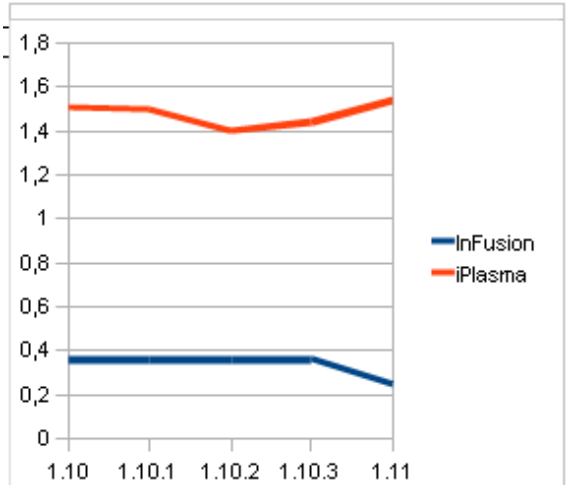

Figure 10: Shotgun Surgery

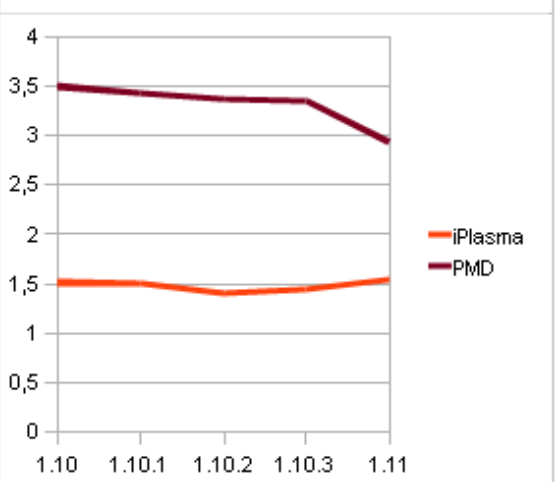

Figure 8: Long Parameter List

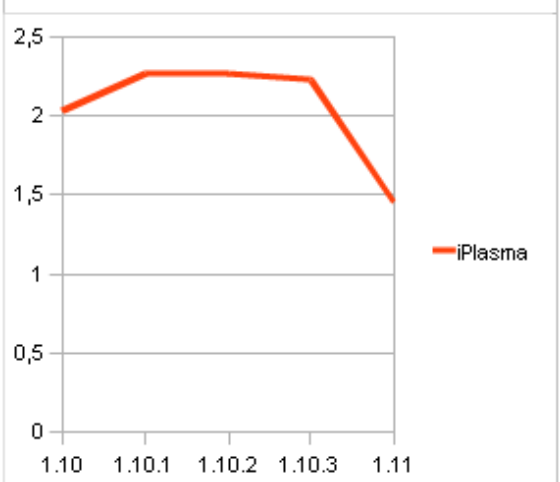

Figure 11: Speculative Generality

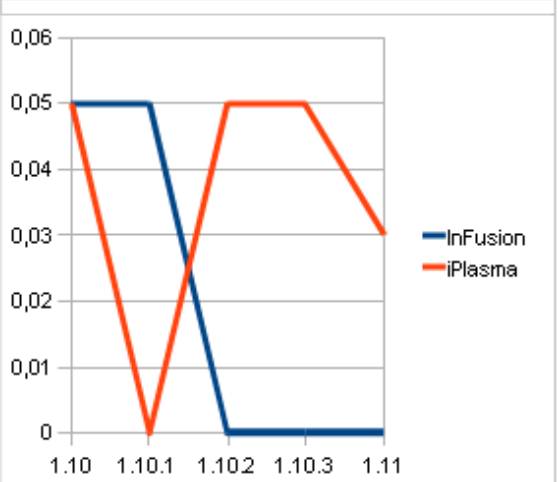

Figure 9: Refused Parent Bequest

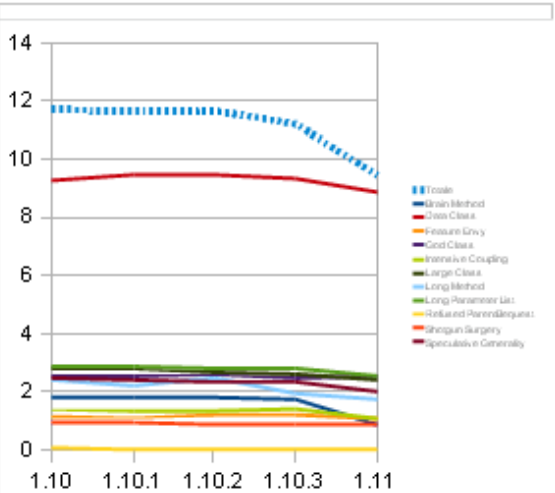

Figure 12: Global System Evolution

Table 6: Brain Method Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Infusion	35 (1,82%)	35 (1,82%)	35 (1,82%)	34 (1,75%)	24 (0,88%)
iPlasma	35 (1,82%)	35 (1,82%)	35 (1,82%)	34 (1,75%)	24 (0,88%)

Table 7: Data Class Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Infusion	38 (9,66%)	39 (9,87%)	39 (9,84%)	39 (9,70%)	50 (9,17%)
iPlasma	35 (8,90%)	36 (9,11%)	36 (9,09%)	36 (8,95%)	47 (8,62%)

Table 8: Feature Envy comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
JDeodorant	18 (0,94%)	8 (0,41%)	17 (0,88%)	11 (0,56%)	2 (0,07%)
Stench Blossom	46 (2,4%)	44 (2,29%)	46 (2,39%)	48 (2,47%)	64 (2,34%)
InFusion	7 (0,36%)	8 (0,41%)	8 (0,41%)	8 (0,41%)	8 (0,29%)
iPlasma	27 (1,41%)	27 (1,40%)	28 (1,45%)	31 (1,60%)	42 (1,54%)

Table 9: God Class comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
JDeodorant	7 (1,78%)	6 (1,51%)	8 (2,02%)	5 (1,24%)	22 (4,03%)
InFusion	11 (2,79%)	11 (2,78%)	11 (2,77%)	11 (2,73%)	13 (2,38%)
iPlasma	10 (2,54%)	10 (2,53%)	10 (2,52%)	10 (2,48%)	13 (2,38%)

Table 10: God Class – Décor's benchmark results on version 1.10.2	
Known God Class	**Detected God Class**
9 (2,27%)	13 (3,28%)

Table 11: Intensive Coupling Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Infusion	27 (1,41%)	27 (1,40%)	27 (1,40%)	28 (1,44%)	29 (1,06%)
iPlasma	26 (1,41%)	26 (1,40%)	26 (1,40%)	28 (1,44%)	29 (1,06%)

Table 12: Large Class Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Stench Blossom	15 (3,81%)	13 (3,29%)	14 (3,53%)	14 (3,48%)	16 (2,94%)
PMD	9 (2,29%)	10 (2,53%)	9 (2,27%)	9 (2,23%)	12 (2,18%)

Table 13: Long Method comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
JDeodorant	164 (8,57%)	124 (6,45%)	186 (9,66%)	69 (3,56%)	57 (2,09%)
Stench Blossom	58 (3,03%)	58 (3,01%)	56 (2,91%)	57 (2,94%)	71 (2,6%)
iPlasma	11 (0,57%)	11 (0,57%)	11 (0,57%)	11 (0,57%)	14 (0,4%)
PMD	47 (2,45%)	47 (2,44%)	45 (2,33%)	46 (2,37%)	55 (2,01%)

Table 14: Long Method – Décor's benchmark results on version 1.10.2	
Known Long Method	**Detected Long Method**
45 (2,33%)	22 (1,14%)

Table 15: Long Parameter List Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
iPlasma	44 (2,3%)	45 (2,34%)	44 (2,28%)	44 (2,27%)	58 (2,12%)
PMD	67 (3,5%)	66 (3,43%)	65 (3,37%)	65 (3,35%)	81 (2,94%)

Table 16: Long Parameter List – Décor's benchmark results on version 1.10.2	
Known Long Parameter List	**Detected Long Parameter List**
54 (2,8%)	43 (2,23%)

Table 17: Refused Parent Bequest Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Infusion	1 (0,05%)	1 (0,05%)	0 (0%)	0 (0%)	0 (0%)
iPlasma	1 (0,05%)	0 (0%)	1 (0,05%)	1 (0,05%)	1 (0,03%)

Table 18: Refused Parent Bequest– Décor's benchmark results on version 1.10.2	
Known Refused Parent Bequest	Detected Refused Parent Bequest
9 (2,27%)	13 (3,28%)

Table 19: Shotgun Surgery Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
Infusion	7 (0,36%)	7 (0,36%)	7 (0,36%)	7 (0,36%)	7 (0,25%)
iPlasma	29 (1,51%)	29 (1,50%)	27 (1,40%)	28 (1,44%)	42 (1,54%)

Table 20: Speculative Generality Comparisons					
	1.10	**1.10.1**	**1.10.2**	**1.10.3**	**1.11.1**
iPlasma	8 (2,03%)	9 (2,27%)	9 (2,27%)	9 (2,23%)	8 (1,46%)