# Micro Patterns in Agile Software

Giulio Concas, Giuseppe Destefanis,
Michele Marchesi, Marco Ortu, and Roberto Tonelli

Department of Electrical and Electronic Engineering (DIEE)
University of Cagliari
Cagliari, Italy
{concas,giuseppe.destefanis,michele,
marco.ortu,roberto.tonelli}@diee.unica.it

**Abstract.** In this paper we present a study on micro patterns in different releases of two software systems developed with Object Oriented technologies and Agile process. Micro patterns are design decisions in code that can be easily automatically recognised. Gil and Maman introduced the concept to support providing objective assessment of design decisions [1]. They catalogued 27 micro patterns that capture a variety of programming practices in Java. Micro patterns can be a useful metrics in order to measure the quality of software by showing that certain categories of micro patterns are more fault prone than others, and that the classes that do not correspond to any category of micro patterns are more likely to be faulty. In our study we present some empirical results on two case studies of systems developed with Agile methodologies, and compare them to previous results obtained for non Agile systems. In particular we have verified that the distribution of micro patterns in a software system developed using Agile methodologies does not differ from the distribution studied in other systems, and that the micro patterns fault-proneness is about the same. We also analyzed how the distribution of micro patterns changes in different releases of the same software system. We demonstrate that there is a relationship between the number of faults and the classes that do not match with any micro patterns. We found that these classes are more likely to be fault-prone than the others even in software developed with Agile methodologies.

**Keywords:** agile, micro pattern, data mining, object oriented programming.

## 1 Introduction

Software quality metrics [20] aim measuring how much a software is good especially from the point of view of being error-free and easy to modify and maintain. Software quality metrics tend to measure whether software is well structured, not too simple and not too complex, with cohesive modules that minimize their coupling. Many quality metrics have been proposed for software, depending also on the paradigm and languages used there are metrics for structured programming, object-oriented programming, aspect-oriented programming, and so on. In this

H. Baumeister and B. Weber (Eds.): XP 2013, LNBIP 149, pp. 210–222, 2013.

paper, we will focus on micro patterns metrics. Micro patterns are design decisions in code that can be easily and automatically recognized. Gil and Maman introduced the concept to support providing objective assessment of design decisions [1]. They catalogued 27 micro patterns that capture a variety of programming practices in Java, from inheritance, to data encapsulation, to the emulation of typical practices of procedural programming. The 27 micro patterns proposed by Gil and Maman were shown by them to be present in 75 % of classes they analyzed. Some of those patterns are regarded as anti patterns [10] representing practices that are considered to be poor design practice although it is important to emphasize that there is no agreement about which micro patterns are considered anti patterns. Thus classes can be divided into 2 categories: MP (Micro Patterns) and NMP (no Micro Pattern) namely those that match one or more of the 27 micro patterns, and those that do not match any micro patterns. Given the purpose of micro patterns, a question naturally arises as to whether there is a relationship between the use of different patterns and the quality of the code. In particular there are no studies investigating the diffusion and the distribution of micro patterns in software systems developed using Agile methodologies [2].

In this work we will present the possible use of micro patterns metrics to indirectly assess the quality of the developed software, by showing the relationship between micro patterns and faults and in this context, we assess the ability of micro patterns to discriminate the usage of Agile practices. We present results on different releases of two software systems on two industrial case-study. We understand that the presented evidence is anecdotal, but with real software projects it is very difficult to plan multi-project researches of this kind. This is because software houses tend to be very secretive about their projects. We hope that other researchers will try to replicate the presented results on similar projects whose data they can access. The target of our research is the evolution of a software project consisting of the implementation of floss-AR, a program to manage the Register of Research of universities and research institutes. floss-AR was developed with a full object-oriented (OO) approach and released with GPL v.2 open source license. The second system is a Web application, which has been implemented through a specialization of an open source software project, jAPS (Java Agile Portal System) [6], that is a Java framework for Web portal creation. This system is certified as a software developed using Agile methodologies.

In order to verify the use of Agile methodologies during the development phases of the analyzed systems, we submitted a questionnaire to the developers such as to have greater knowledge about Agile methodologies used.
We decided to organize our paper answering to the following research questions:

- **RQ1:** Do software systems developed with Agile methodologies have a different distribution of micro patterns with respect to non Agile open source systems?
- **RQ2:** Is the micro patterns faults-proneness the same for Agile and non Agile software?
- **RQ3:** Does the micro patterns distribution change during software evolution? If yes, how?

## 2 Related Works

After the work of Gil and Maman that defines the catalog of the micro patterns [1], several works have appeared in this field. Arcelli and Maggioni suggest a novel approach for the detection of micro patterns which is aimed at identifying types that are very close and similar to a correct micro patterns implementation, even if some of the methods and/or attributes of the type do not comply with the constraints defined by the micro patterns [4]. The new interpretation is based on the number of attributes (NOA) and the number of methods (NOM) of a type. Similar studies to those discussed in our work have been conducted for design patterns [5]: Heuzeroth et al. presented an approach to support the understanding of software systems by detecting design patterns automatically using static and dynamic analyses [7]. Aversano et al. report an empirical study showing that for three open source projects, the number of defects in design-pattern classes is in several cases correlated with the scattering degree of their induced crosscutting concerns, and also varies among different kinds of patterns [8]. Destefanis et al. [3] analyzed the relationship between faults and the remaining 25% of classes that do not match with any micro pattern. They found that these classes are more likely to be fault-prone than the others. Tasharofi et al. [14] provide a set of high-level process patterns for Agile development which have been derived from a study of seven Agile methodologies based on a proposed generic Agile Software Process. These process patterns can promote method engineering by providing classes of common process components with can be used for developing, tailoring, and analyzing Agile methodologies. Concas et al. in [13] studied and discussed the evolution of the classical software metrics and their behavior related to the Agile practices adoption level. The authors show that, in the reported case study, a few metrics are enough to characterize with high significance the various phases of the project. Consequently, software quality, as measured using these metrics, seems directly related to Agile practices adoption.

## 3 Methodology

The goal of this paper is to investigate the possible relationship between Agile methodologies and micro patterns. We submitted to the developers of the floss-AR software system, a questionnaire in order to evaluate the effective use of Agile methodologies in the early stages of software development [17]. We developed a custom Java tool, based on Gil and Maman's research [1] in order to extract from the software systems analyzed the data relative to the micro patterns distribution. We tested our tool on the data-set used in [1] finding the same results. The tool works in two steps:

- the first step consists in parsing the source code and in generating a series of files containing information relative to the various classes, fields, methods, calls and so on;
- in the second step the tool calculates the presence of the micro patterns for each class of the analyzed system, using the files produced in the first step.

The tool uses the definitions given by Arcelli and Maggioni described in [4]. The class is assigned to only one micro pattern, the one with the highest GSR (Global Similarity Ratio). GSR is a real number between zero (complete absence of the micro pattern) and one (presence of the micro pattern as defined in [1]). Intermediate values indicate a partial presence of the micro pattern. Each software system analyzed is characterized by a GSR matrix where each row represents the value for a class and each column contain a GSR value for each micro pattern. The correlation between columns of the GSR matrix provides important information about the relationship between different micro patterns, for example if the matching of one micro pattern with a class implies the matching of an other micro pattern with the same class. We analyzed the two systems developed using Agile methodologies and we have studied the distribution and the evolution of micro patterns through different releases.

The micro patterns catalog contains several categories that in the literature are considered like anti patterns [12] as descriptive of bad programming practices not related to the object orientation techniques.

In [3] Destefanis et al. show that there are other micro patterns categories prone to fault and that the classes of a software system that does not belong to any category of micro patterns are more prone to faults. In this paper we analyzed the different releases of the floss-AR system in order to verify if:

- also in this case there is a relationship between the number of faults and anti micro patterns;
- there is a relationship between number of faults and micro patterns more fault prone;
- there is a relationship between number of faults and classes that do not belong to any micro patterns category.

The analysis cannot have statistical significance (because it is performed on a single system), but it is however interesting and a good starting point to further studies. To establish the link between source code and fix operation we adopt the traditional heuristics proposed by Bachmann and Bernstein [11]:

1. Scan through the change logs for bug report in a given format (e.g. fix bug, fix issue and so on).
2. Exclude all false-positive bug numbers (e.g. r420, 2009-05-07 10:47:39 -0400 and so on).
3. Check if there are other potential bug number formats or false positive number formats, add the new formats and scan the change logs iteratively.
4. Check if potential bug numbers exist in the bug- tracking database with their status marked as fixed.

Based on these heuristics we mine the source code repository (such as CVS and SVN) for commit that fixed a bug. Knowing how many time a class have been debugged and knowing the micro patterns associated (if any) to the class we could then evaluate the fault proneness of micro patterns for the system analyzed.

## 4 Results

In this section we present the results of the survey to developers and on the analysis performed on the source code of the Agile systems. In particular we show how the Agile development impacts on the micro patterns statistics, and on the fault proneness of micro patterns, anti patterns and the set composed by the classes that do not match with andy micro patterns of the catalog (no micro patterns category: NMP).

### 4.1 Survey

The results of the survey clearly show that Agile development has been applied for the floss-AR system. Tabs. 1 2 3 resume the survey's results.

**Table 1.** floss-AR developers survey (5 developers)

| Question | Very good | Good | Discrete | Adequate | Not adequate |
|---|---|---|---|---|---|
| How would you describe the collaboration of the team? | 4 | 1 | 0 | 0 | 0 |

**Table 2.** floss-AR developers survey (5 developers)

| Question | Yes | No |
|---|---|---|
| The collaboration inside the team increased the productivity? | 5 | 0 |
| Did you take part in developing the whole system? | 3 | 2 |
| Do you have favourite programming styles? | 2 | 3 |
| Have the project decisions been discussed together with the team? | 5 | 0 |
| Did you interact directly with the customer? | 4 | 1 |
| Did you use refactoring? | 5 | 0 |

The questions are divided in three groups according to the format of the possible answers. The first question requires an answer with 5 possibilities, in the second set the questions are posed in a YES or NO form, while in the third set the questions require a short sentence answer.

For developing floss-AR the following Agile practices have been applied:

- Pair programming
- Stand Up Meeting
- Refactoring
- On Site Customer

According to further discussions with the developers team, we are also able to identify four main phases of development:

**Table 3.** floss-AR developers survey (5 developers)

| Question | Answer |
|---|---|
| Which Agile methodologies did you use during development? | • Pair Programming<br>• Stand Up Meeting<br>• Refactoring<br>• On Site Customer |
| How often did you interact with the customer? | 1-2 times per month |
| How often did you use refactoring? | 2-3 times per month |

- Phase 1 (Initial Agile): a phase characterized by the full adoption of all practices, including testing, refactoring and pair programming. This is the phase leading to the implementation of a key set of the system features. In practice, specific classes to model and manage the domain of research organizations, roles, products, and subjects were added to the original classes managing the content management system, user roles, security, front end and basic system services. The new classes include service classes mapping the model classes to the database, and allowing their presentation and user interaction.
- Phase 2 (Cowboy Coding): this is a critical phase, characterized by a minimal adoption of pair programming, testing and refactoring, because a public presentation was approaching, and the system still lacked many of the features of competitors' products. So, the team rushed to implement them, compromising the quality.
- Phase 3 (Refactoring): an important refactoring phase, characterized by the full adoption of testing and refactoring practices and by the adoption of a rigorous pair programming rotation strategy. The main refactorings performed were Extract Superclass, to remove duplications and extract generalized features from classes representing research products, and corresponding service classes, and Extract Hierarchy applied to a few big classes, such as an Action class that managed a large percentage of all the events occurring in the user interface. This phase was needed to fix the bugs and the bad design that resulted from the previous phase.
- Phase 4 (Mature Agile): Like Phase 1, this is a development phase characterized by the full adoption of the entire set of practices, until the final release.

### 4.2 Source Code Analysis

We next report the results on how Agile methodologies can impact on the micro patterns distribution and on the fault proneness of the code. In Tabs. 4 5 we report the micro patterns distributions for each release of the floss-AR and Japs systems, in order to show how such distributions evolve from one release to the next.

**Table 4.** jAPS micropattern distribution (%)

| MP | 1.0 | 1.2 | 1.4 | 1.6 | 1.6.2 | 1.8 | 1.8.2 | 2.0 |
|---|---|---|---|---|---|---|---|---|
| DESIGNATOR | 2.14 | 1.79 | 2 | 3.3 | 3 | 4.32 | 6.83 | 9.6 |
| TAXONOMY | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| POOL | 0 | 0 | 0 | 0.55 | 0.54 | 0.27 | 0 | 0.35 |
| JOINER | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FUNCTIONPOINTER | 27.1 | 23.3 | 27.5 | 18.7 | 19.5 | 18.1 | 16.7 | 7.18 |
| FUNCTIONOBJECT | 0.71 | 6.1 | 0 | 2.2 | 2.7 | 1.89 | 2.02 | 1.22 |
| COBOLLIKE | 0 | 0 | 0 | 0.27 | 0.27 | 0.81 | 0.75 | 0.5 |
| STATELESS | 0.71 | 0 | 1 | 0.82 | 0.82 | 1.08 | 1.01 | 1.22 |
| COMMONSTATE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.17 |
| IMMUTABLE | 0 | 3.2 | 0 | 0.82 | 0.82 | 0.81 | 0.75 | 0.87 |
| RESTRICTEDCREATION | 0.35 | 0.4 | 0.33 | 0.55 | 0.54 | 0.54 | 0.5 | 0.17 |
| SAMPLER | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| BOX | 4.64 | 15.4 | 3.98 | 0.27 | 0.27 | 0.27 | 0.25 | 1.4 |
| COMPOUNDBOX | 7.5 | 10 | 12.3 | 7.1 | 17.9 | 7.02 | 6.83 | 11.9 |
| CANOPY | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RECORD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DATAMANAGER | 0.35 | 0.35 | 0 | 0 | 0 | 0 | 0 | 0 |
| SINK | 15.3 | 3.9 | 15.6 | 4.14 | 3.5 | 2.7 | 2.78 | 2.45 |
| OUTLINE | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0.35 |
| TRAIT | 0 | 0 | 0 | 0 | 0 | 0 | 1.3 | 1.1 |
| STATEMACHINE | 0.71 | 0 | 0.66 | 0.82 | 0.82 | 0.54 | 0.5 | 5.4 |
| PURETYPE | 0 | 0 | 0 | 0 | 0.5 | 0.8 | 0.3 | 0.2 |
| AUGMENTEDTYPE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PSEUDOCLASS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| IMPLEMENTOR | 0 | 0.71 | 0.3 | 0.27 | 0.27 | 0.27 | 0.25 | 0.35 |
| OVERRIDER | 0 | 0 | 0.3 | 0.82 | 0.82 | 0.54 | 0.5 | 0.87 |
| EXTENDER | 25 | 27.9 | 27.5 | 36.1 | 35.9 | 37.2 | 34.6 | 25.1 |
| TOTAL | 84 | 73 | 85.7 | 77 | 76.6 | 76.4 | 74.4 | 75.1 |

Both systems respect the Gil and Maman statement that about 75% of classes belong to at least one micro pattern. This means that micro patterns are good descriptors also for software developed with Agile methodologies. The distributions of micro patterns among classes roughly respect the same proportions found for software developed with traditional methodologies [3]. In fact previous results show that Extender, Sink and Function Pointer are the most common micro patterns, while Taxonomy, Pool, Sampler and Record are almost absent. One key point is the behavior of anti patterns, which are indicators of bad programming practices [19]. The overall anti patterns behavior is captured by Function Pointer, because classes belonging to others anti patterns, like Pool or Record, are a very small fraction of the total number of classes. Such behavior is displayed in Fig.1 (left side), which shows an overall decreasing trend in the usage of anti patterns. This suggests that the constant application of Agile methodologies during software development across different releases may impact

**Table 5.** floss-AR micro patterns distribution (%)

| MP | CA | SAR | SS | OS | 2.1.1 |
|---|---|---|---|---|---|
| DESIGNATOR | 1.5 | 1.5 | 1.6 | 1.38 | 0.9 |
| TAXONOMY | 0 | 0 | 0 | 0 | 0 |
| POOL | 0.2 | 0.2 | 0.36 | 0.3 | 0.76 |
| JOINER | 0 | 0 | 0 | 0 | 0 |
| FUNCTIONPOINTER | 20.2 | 19.7 | 22.8 | 17.8 | 13.31 |
| FUNCTIONOBJECT | 2.5 | 2.4 | 2 | 4.45 | 1.53 |
| COBOLLIKE | 0.17 | 0.17 | 0.14 | 0.46 | 0.13 |
| STATELESS | 0.4 | 0.3 | 0.29 | 1.07 | 2.57 |
| COMMONSTATE | 0.2 | 0.2 | 0.14 | 0.15 | 0.06 |
| IMMUTABLE | 0.2 | 0.2 | 0.14 | 0.76 | 0.06 |
| RESTRICTEDCREATION | 0.1 | 0.1 | 0.29 | 0.30 | 0.06 |
| SAMPLER | 0 | 0 | 0 | 0 | 0 |
| BOX | 2 | 2 | 3.21 | 0.15 | 13.79 |
| COMPOUNDBOX | 7.9 | 8.2 | 7.45 | 10.4 | 12.61 |
| CANOPY | 0 | 0 | 0 | 0 | 0 |
| RECORD | 0 | 0.2 | 0 | 0.2 | 1.6 |
| DATAMANAGER | 0 | 0 | 0 | 1.68 | 1.74 |
| SINK | 18.9 | 18.6 | 17.2 | 3.53 | 14.77 |
| OUTLINE | 0 | 0 | 0 | 0.3 | 1.1 |
| TRAIT | 0.33 | 0.3 | 0.29 | 1.2 | 0.13 |
| STATEMACHINE | 0.17 | 0.17 | 0.29 | 0.15 | 0.06 |
| PURETYPE | 0 | 0 | 0 | 0.3 | 0.1 |
| AUGMENTEDTYPE | 0 | 0 | 0 | 0 | 0 |
| PSEUDOCLASS | 0 | 0 | 0 | 0 | 0 |
| IMPLEMENTOR | 1.7 | 1.22 | 1.46 | 2.61 | 0.69 |
| OVERRIDER | 0.33 | 0.34 | 0.29 | 1.07 | 0.2 |
| EXTENDER | 28.4 | 28.8 | 27.7 | 28.4 | 16.58 |
| TOTAL | 85.1 | 84.8 | 85.8 | 75.5 | 81.6 |

positively the software quality, carrying as side effect the reduction in the use of bad programming practices.

### 4.3 Micro Patterns and Faults

Next we examine the relationship among micro patterns and faults in the floss-AR releases. The top part of Tab. 6 shows the distribution of faulty classes among non micro patterns (NMP) and micro patterns (MP). It must be noted that NMP classes are only 25% of the total classes, and nevertheless they own the larger percentage of faulty classes, except for the last release, where the percentage of faulty classes is the same as the percentage of NMP in the entire release. This result for the first four releases is in agreement with those reported in [3], where NMP own most of the faults. This means that software developed through the adoption of Agile methodologies does not differ from other software

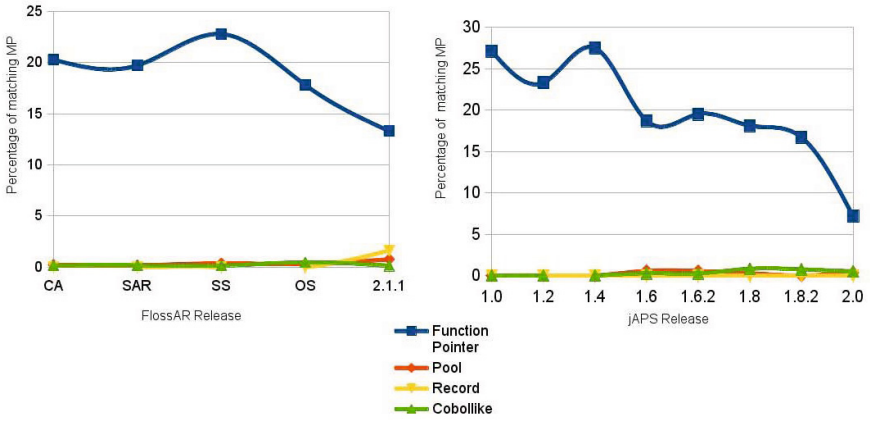


**Fig. 1.** Left side: floss-AR - Right side: Japs

**Table 6.** FlossAR fault-prone analysis

| | | OS(%) | CA(%) | SAR(%) | SS(%) | 2.1.1(%) |
|---|---|---|---|---|---|---|
| Distribution of faulty classes | NMP | 63.12 | 62.41 | 71.63 | 70.92 | 23.4 |
| among NMP and MP | MP | 36.87 | 37.58 | 28.36 | 29.07 | 76.59 |
| Percentage of MP faults | | | | | | |
| Fault Percentage of AMP | | 12.76 | 12.05 | 7.8 | 7.8 | 23.4 |
| Fault Percentage of fault-prone MP faults | | 18.43 | 14.89 | 11.34 | 13.47 | 32.62 |
| Fault Percentage of other MP | | 5.67 | 10.63 | 9.21 | 7.8 | 20.56 |

with respect to such distribution. The result for the last release is somehow unexpected, and we cannot explain it with the data at our disposal. Further analysis are needed in order to understand the reasons for this inversion in the fault proneness.

The bottom part of Tab. 6 shows how faults are distributed among the different MP categories: anti micro patterns (AMP), fault-prone MP, and other MP, where fault prone MP are identified by the analysis performed in [3]. Also in this case the total percentage of faulty classes in the last release is different than in previous releases, but the distribution among AMP, fault prone MP, and other MP is again respected. These results confirm that also in Agile systems the most fault prone micro patterns are Extender and Compound Box, and that also the AMP classes are more fault prone than others.

### 4.4 Discussion

According to these results, we can now answer to the research questions:

**RQ1: Do software systems developed with Agile methodologies have a different distribution of micro patterns with respect to non Agile open source systems?**

The answer to this research question is negative. According to tabs. 4, 5, the distributions of classes across micro patterns is roughly the same described in [3], where 8 systems were analyzed. They are very similar for both Japs and floss-AR, in all the releases analyzed. This result suggests that the use of Agile methodologies and programming practices does not influence the distribution of micro patterns in the classes.

**RQ2: Is the micro patterns faults-proneness the same for Agile and non Agile software?**

The answer to this question is positive except for the last release of floss-AR. Comparing the results obtained for the first 4 releases of floss-AR analyzed (Tab. 6, top part) NMP classes are by far the most fault prone classes. The more detailed analysis reported in Tab. 6 (bottom part) shows that among the classes matching with at least one micro pattern the Extender and Compound box micro patterns as well as the anti patterns are the most fault prone. This result confirms the findings reported in [3] and shows that the fault prone micro patterns distributions in Agile software is similar to the one found in systems developed without the adoption of Agile methodologies.

**RQ3: Does the micro patterns distribution change during software evolution? If yes, how?**

The answer to this research question is not univocal. In general we have shown that across all the releases the micro patterns distribution remains the same, with the exception of the anti patterns classes. In fact we found a decrease of the percentage of anti patterns classes in both systems across the releases. This may be related to the continuous adoption of Agile methodologies during development and maintenance.

## 5 Threats to Validity

Threats to construct validity are related to the Agile methodologies not used during the system's development (like TDD and continuous integration). This may influence our conclusion that the use of agile methodologies may improve software quality, given that agile development has been adopted partially. Another threat to construct validity is related to the relationship between micro patterns and faults. We assume, based on previous works, that MP are related to software defectiveness. This result has not been generalized to all software systems, thus not necessarily the micro patterns catalogue is directly related to software defectiveness. Nevertheless we believe that our work can build a first step in this direction. Threats to internal validity are related to the fact that with different values of micro patterns could be possible to observe different correlations. Threats to external validity are related to generalization of our

conclusions. With regard to the system studied in this work we considered only open source systems written in Java, and this could affect the generality of the discussion and thus our results are not representative of all environments or programming languages. Commercial software is typically developed using different platforms and technologies, with strict deadlines and cost limitation, and by developers with different experiences. This might result in different micro patterns distributions, which is another threat for the external validity. Another threat regards the relationships among anti patterns and faults, which has been studied only for the floss-AR system. Finally we have another threat to conclusion validity: there is not an estimated error on the recognition of a particular micro pattern for a given class.

## 6 Conclusions

The goals of this research were the analysis of micro patterns distribution in Agile open source software and the analysis of the relationship between MP-NMP and faulty classes. We used the Java tool discussed in [3] in order to extract the data relative to the micro patterns distribution in the two Agile software system studied.

For the floss-AR system we analyzed the change log for bug report and extracted fix operation according to the traditional heuristic proposed in [11]. We also submitted to the floss-AR developers team a questionnaire in order to evaluate the effective use of Agile methodologies, while for Japs this is certified on the web site [6].

Our analysis shows that the micro pattern distribution among classes is the same for the two systems, and remains roughly the same as the one found in non agile systems. Thus the adoption of agile methodologies does not influence such distribution. For example, Gil and Maman statement's that about 25% of classes does not match with any micropattern, is confirmed also in the two agile systems analyzed, for all the releases.

The analysis of fault prone classes shows that in agile systems the Extender and Compound box micro patterns are fault prone, as well as the AMP classes. In particular the most fault prone classes are those not belonging to any micro pattern. The last release of floss-AR represents an exception to this rule, even if the percentage of faulty classes belonging to NMP (23.4%), is still larger than the percentage of NMP classes in all the systems (18.4%).

Finally we found that the micro patterns distribution across the releases is unchanged, with the exception of the anti pattern classes, which displays a decreasing trend.

We can conclude that micro patterns may be helpful to evaluate the quality of an Agile software project during the development process. A tool like the one used in the present work could be used in order to monitor the different stages of development, and possibly to control the temporal evolution of each category of micro patterns. It can be seen from our empirical results that classes that do not correspond to any micro patterns are more fault-prone and this supports

that the use of a design methodology increases the quality of the code. Considering the natural adaptiveness of Agile development it could be useful to monitor the evolution of the most fault-prone micro patterns in order to increase the software quality and decrease the amount of defects.

**Acknowledgment.** This research is supported by Regione Autonoma della Sardegna (RAS), Regional Law No. 7-2007, project CRP-17938 LEAN 2.0

## References

1. Gil, J.Y., Maman, I.: Micro pattern in Java Code. In: Proceedings of the 20th Object Oriented Programming Systems Languages and Applications, San Diego, CA, USA, p. 97116 (2005)
2. Agile Manifesto, `http://www.agilemanifesto.org`
3. Destefanis, G., Tonelli, R., Tempero, E., Concas, G., Marchesi, M.: Micro Pattern Fault-Proneness. In: 2012 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 302–306. IEEE (September 2012)
4. Arcelli, F., Maggioni, S.: Metrics-based Detection of Micro pattern to improve the Assesment of Software Quality. In: Proceedings of 1st Symposium on Emerging Trends in Software Metrics (ETSM 2009), Italy (May 2009)
5. Gamma, E., Helm, R., Jhonson, R., Vlissides, J.: Design Pattern: Elements of Reusable Object-Oriented Software. Addison Wesley (1995)
6. JAPS: Java agile portal system, `http://www.japsportal.org`
7. Heuzeroth, D., Holl, T., Hogstrom, G., Lowe, W.: Automatic Design Pattern Detection. In: IWPC 2003 Proceedings of the 11th IEEE International Workshop on Program Comprehension (2003)
8. Aversano, L., Cerulo, L., Di Penta, M.: Relationship between design pattern defects and crosscutting concern scattering degree: an empirical study. IET Softw. 3(5), 395–409 (2009)
9. Dorairaj, S., Noble, J., Malik, P.: Understanding Team Dynamics in Distributed Agile Software Development. In: Wohlin, C. (ed.) XP 2012. LNBIP, vol. 111, pp. 47–61. Springer, Heidelberg (2012)
10. Bloch, J.: Effective Java Programming Language Guide. Addison-Wesley (June 2011)
11. Bachmann, A., Bernstein, A.: Software process data quality and characteristics: a historical view on open and closed source projects. In: IWPSE-Evol 2009 Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops. ACM (2009)
12. Destefanis, G., Tonelli, R., Concas, G., Marchesi, M.: An analysis of anti micro patterns effects on fault proneness in large Java systems. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 1251–1253. ACM (March 2012)
13. Concas, G., Marchesi, M., Destefanis, G., Tonelli, R.: An empirical study of software metricsfor assessing the phases of an agile project. International Journal of Software Engineering and Knowledge Engineering 22, 525–548 (2012)

14. Tasharofi, S., Ramsin, R.: Process Patterns for Agile Methodologies. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences. IFIP, vol. 244, pp. 222–237. Springer, Boston (2007)
15. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR, Upper Saddle River (2003)
16. Empirical studies of agile software development: A systematic review. Tore Dyba, Torgeir Dingsoyr. SINTEF ICT, S.P. Andersensv. 15B, NO-7465 Trondheim, Norway
17. Hoda, R., Noble, J., Marshall, S.: How much is just enough?: some documentation patterns on Agile projects. In: Proceedings of the 15th European Conference on Pattern Languages of Programs, EuroPLoP 2010, Article 13, 13 pages. ACM, New York (2010)
18. Martinez, J., Diaz, J., Perez, J., Garbajosa, J.: Software Product Line Engineering Approach for Enhancing Agile Methodologies. In: Abrahamsson, P., Marchesi, M., Maurer, F. (eds.) XP 2009. LNBIP, vol. 31, pp. 247–248. Springer, Heidelberg (2009)
19. Bloch, J.: Effective Java Programming Language Guide. Addison-Wesley (June 2011)
20. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6), 476–493 (1994)