Software Metrics in Agile Software: an Empirical Study

Giuseppe Destefanis[1], Steve Counsell[2], Giulio Concas[1], and Roberto Tonelli[1]

[1] DIEE, University of Cagliari, Italy
|giuseppe.destefanis|concas|roberto.tonelli|@diee.unica.it
[2] Brunel University, Kingston Lane, Uxbridge, UK
steve.counsell@brunel.ac.uk

Abstract. This paper presents a software metrics analysis of eight object-oriented systems. Five systems had been developed using Agile methodologies and three using plan-driven methodologies; three systems were written in Python and five in Java. For each system, we considered 10 traditional metrics such as LOC and the Chidamber and Kemerer metrics. These metrics were computed at class level. In our study we present empirical results considering systems developed with Agile methodologies and we compare them with previous results for non Agile systems. In particular, we verify that the distributions of software metrics in a software system developed using Agile methodologies does not differ from the distribution in systems developed using plan-driven methodologies.

Key words: agile, software metrics, data mining, object-oriented programming

1 Introduction

Software engineers have been trying to measure software to gain quantitative insights into its properties and quality since its inception. IEEE defines Software Engineering as the application of a "systematic, disciplined, quantifiable approach to the development, operation and maintenance of software." With the advent of the object-oriented (OO) approach, specific measures have been introduced to evaluate the quality of software systems. The rationale behind OO metrics is that a good design must keep complexity low, and this can be accomplished by minimising coupling and increasing cohesion. The first attempt in this direction was Chidamber and Kemerer's metrics suite (CK), and these have became the most popular OO metrics suite and the process of defining new measures is still a vibrant research area; that said, theoretical reasons supporting the adoption of specific metrics is not enough. Software engineers need to have empirical evidence that these metrics are actually related to software quality. Unfortunately, "software quality" is an elusive concept, software being an immaterial entity that cannot be physically measured in traditional ways. In general software quality has many different meanings, it is associated with practices that lead to software products that are accurate, effective, delivered on time and within budget.

It is still difficult to relate software metrics to the phenomena we want to improve and to reduce the complexity of software, new development methodologies and tools are being introduced.In particular, to our knowledge, there are no studies investigating the distribution of traditional metrics in software developed using Agile methodologies. [19].

In this paper, we attempt to present the results about software metrics distributions on 8 open source software system, (5 developed using Agile methodologies, 3 developed using plan-driven methodologies, 5 developed using Java, 3 developed using Python) in order to study possible differences due to Agile methodologies. We considered the following systems:

- **FlossAR** [4]: a program to manage the Register of Research of universities and research institutes. floss-AR was developed with a full OO approach and released with GPL v.2 open source license.
- **jAPS (Java Agile Portal System)** [5] is a Web application, implemented through a specialization of an open source software project that is a Java framework for Web portal creation. This system is certified as a software developed using Agile methodologies.
- **OpenErp** [7]: OpenERP is an open-source enterprise resource planning (ERP) software actively programmed, supported, and organized by OpenERP s.a. OpenERP similar to many open source projects where customized programming, support and other services are also provided by an active global community and partner network. OpenERP is developed using agile software development and test-driven development methodologies.
- **OpenBravo** [6]: Openbravo ERP is a web-based ERP business solution for small and medium sized companies released under the Openbravo Public License, the program is among the top ten most active projects of Sourceforge as of January 2008. OpenBravo is also known as Openbravo Agile Erp.
- **Zope** [8]: Zope is a community project concerned with free and open-source, OO web application server written in the Python programming language. Zope stands for "Z Object Publishing Environment" and was the first system using the now common object publishing methodology for the Web. Zope has been recognized as a Python killer app, an application that helped put Python in the spotlight. The Zope project pioneered the practice of sprints for open source software development. Sprints are intensive development sessions when programmers, often from different countries, gather in one room and work together for a couple of days or even several weeks. During the sprints, various practices drawn from agile software development are used, such as pair programming and test-driven development
- **Ant** [1]: Apache Ant is a software tool for automating software build processes. It is similar to Make but is implemented using the Java language, requires the Java platform and is best suited to building Java projects.
- **Weka (Waikato Environment for Knowledge Analysis)** [3] is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. Weka is free software available under the GNU General Public License.

– **Blender** [2]: Blender is a free and open-source 3D computer graphics software product used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games.

OpenERP and OpenBravo are both ERP systems developed using Agile methodologies but written using different languages: OpenERP in Python, OpenBravo in Java (information about the use of Agile methodologies are available on their respective websites [7], [6]). This fact is interesting for evaluating differences between programming languages. For Japs and Zope it is possible to check information about Agile methodologies during development phases on respective websites. Regarding to floss-AR system, according to further discussions with the developers team, the following Agile practices have been applied:

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

Pair Programming is an Agile Software development technique in which two developers work together at one workstation. The driver writes code while the observer reviews each line of code. A Stand Up Meeting is a daily team meeting held to provide a status update to the team members. Refactoring is the process of restructuring an existing body of code altering its internal structure without changing its external behavior. On Site Customer describes the need to have on-site access to people who have the authority and ability to provide information pertaining to the system being developed and to make pertinent and timely decisions regarding the requiriments. Results presented in this paper are not conclusive because we have analyzed only open source software; the link between open source development and the Agile methodology is under-researched - Adams et al. [10] studied gaps between Agile development and open source development. This work is a starting point and clearly, further research is needed to prove and validate differences between metrics distributions generated by Agile methodologies and those generated by plan-driven methodologies, especially considering proprietary software developed by companies.

In this paper, we answer the following research questions:

- **RQ1:** Is it possible to recognize the use of Agile methodologies through the analysis of software metrics?
- **RQ2:** Metrics distributions generated from software developed using Agile methodologies are similar to metrics distributions generated from software developed using plan-driven methodologies?
- **RQ3:** It is possible to assert that metric distributions generated from Agile methodologies are related to better quality software?

All data considered in this paper are available online at http://agile.diee.unica.it/xp2014.

2 Related Work

Several studies have analyzed software metric distributions with regard to study software quality and to define a methodology for guiding the softare process development, but to our knowledge no studies have tried to analyze the relationship between software metrics and software development methodologies as Agile processes. Therefore many empirical studies have been performed to validate empirically the CK suite from these two aspects, showing an acceptable correlation between CK metrics values and software fault-proneness and difficulty of maintenance [11], [12], [13], [14]. Other OO metrics suites have also been proposed, MOOD [17] and by Lorenz and Kidd [18], but the CK suite is by far the most popular.

In Adams et al. [9] it is argued that the impact of certain Agile practices (in this case, specifically sprinting) on a Free Software project can be partially assessed through analysis of code repository logs, using average commits per day as a metric. In the paper, sprints from two Free Software projects (Plone and KDE PIM) are assessed and two hypotheses are formulated: do sprints increase productivity? Are Free Software projects more productive after sprints compared with before? The primary contribution of the paper is to show how sprinting creates a large increase in productivity both during the event and immediately after the event itself: this argues for more in-depth studies focussing on the nature of sprinting.

Adams et al. in [10] argue that it is possible to quantify the level of agility displayed by Open Source projects. An indicator of agility, the Mean Developer Engagement (MDE) metric is introduced and tested through the analysis of public project data. Projects sampled from two repositories (KDE and SourceForge) are studied and a hypothesis is formulated: projects from the two samples display a similar level of MDE. The paper provides two main contributions: first, the MDE metric is shown to vary significantly between the KDE and SourceForge projects. Second, by combining MDE with a project's lifespan, it is also shown that SourceForge projects have insufficient uptake of new developers resulting in more active, shorter, initial activity and in a quicker "burning out" of the projects.

Concas et al. [15] present an extensive analysis of software metrics for 111 OO systems written in Java. For each system, authors considered 18 traditional metrics such as LOC and CK metrics, as well as metrics derived from complex network theory and social network analysis; they also considered two metrics at system level, namely the total number of classes and interfaces and the fractal dimension. They discuss the distribution of these metrics and their correlation both at class and system level. They found that most metrics followed a leptokurtotic distribution. Only a couple of metrics have patent normal behaviour while three others are very irregular and even bimodal.

In Concas et al. [35] the authors present a comprehensive study of an implementation of the Smalltalk OO system, one of the first and purest OO programming environment, searching for scaling laws in its properties. They studied ten system properties, including the distributions of variable and method names,

inheritance hierarchies, class and method sizes, system architecture graph. They systematically found Pareto or log-normal distributions in these properties. Programming activity, even when modelled from a statistical perspective, can not be modelled as a random addition of independent increments with finite variance; it exhibits strong organic dependencies on what has been already developed. There is a comparison of the results with similar results obtained for large Java systems. The work shows how the Yule process is able to stochastically model the generation of several of the power-laws found, identifying the process parameters and comparing theoretical and empirical tail indexes. The authors discuss how the distributions found are related to existing OO metrics such as CK's and how they could provide a starting point for measuring the quality of a whole system, versus that of single classes. In fact, the usual evaluation of systems based on mean and standard deviation of metrics can be misleading.

3 Methodology

The aim of this paper is to investigate possible relationship between software metrics obtained from software developed using Agile methodologies and software developed using plan-driven methodologies. Our corpus includes: OpenERP, OpenBravo, Zope, Japs, Ant, Weka, FlossAR, Japs.

We built the software graph of each project analyzing the source code, where the nodes of the graph are associated to the classes of the system. Using this graph, we computed 10 metrics on each node of the graph. We designed and wrote all the code for building the software graph and for computing on it all the considered metrics. The statistical analysis on the results were performed using R.

- IFANIN: Number of immediate base classes;
- NOC: Number of Children (CK): number of directed subclasses of the class;
- NIM: Number of instance methods, methods defined in a class that are only accessible through an object of that class;
- NIV: Number of instance variables, variables defined in a class that are only accessable through an object of that class;
- WMC: Weigthed methods per class (CK), a weighted sum of all the methods defined in a class.
- RFC: Response For a Class (CK): the sum of the number of methods defined in the class and the cardinality of the set of methods called by them and belonging to external classes.
- LOC: Lines of code of the class, excluding blank lines and comments.
- CLOC: Lines of comments of the class.
- NOfS: number of declared statement;
- DIT: Depth of Inheritance Tree (CK), length of longest path from a given class to the root class in the inheritance hierarchy.

For each metric, we computed five statistics on all the classes of a system, aimed at giving a global measure of the value of the metrics for all the classes of the system. These statistics are:

- Mean: the mean of the metric;
- Median: the median of the metric;
- First Quartile: lower quartile = 25th percentile (splits off the lowest 25% of data from the highest 75%);
- Third Quartile: upper quartile = 75th percentile (splits off the highest 25% of data from the lowest 75%);
- Standard deviation of the metric.

We calculated the complementary cumulative distribution function (CCDF) for each metric to assess the distribution type. All CCDF representations of metrics considered in this paper are available at http://agile.diee.unica.it/xp2014. We have reported the most interesting in the results section.

4 Results

Table 1, shows caratheristics of our corpus in terms of development language, Agile methodologies (Yes if used, No if not used), number of classes. In the high part of the table there are systems developed using plan-driven methodologies, in the lower part of the table are systems developed using Agile methodologies (we mantain this convention in the other tables of the paper).

System	Language	Agile	# of Classes
Ant	Java	No	1670
Blender	Python	No	2276
Weka	Java	No	1934
FlossAR	Java	Yes	1441
Japs	Java	Yes	456
OpenBravo	Java	Yes	1513
OpenERP	Python	Yes	1741
Zope	Python	Yes	6852

Table 1: Corpus description

The corpus is homogeneus enough considering the size (number of classes) of analyzed systems, except for Japs and Zope. The first has 456 classes and the latter 6852 classes. This is a desired work set in order to test hypothesis considering two outliers. As may be seen below (tabs. 2,4,5,6) there is no evident relationship between number of classes and the average value of metrics considered. The first result is related to metrics differences between systems developed in Java and systems developed in Python. Python is a general-purpose, high-level OO programming language. Its design philosophy emphasizes code readability and

its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Table 2 shows LOC statistics from the analyzed systems and the average LOC (per class) of systems developed using Python are significantly lower than systems developed in Java (regardless of the development methodology used). It is well known that the LOC metric is well related with code defects [23], and the Python language allows the developer to mantain a low level of LOC. Blender has 38.14 as average LOC value against 80.87 of Ant and 130,9 of Weka. Considering the use of Agile methodologies tab. 2 shows that there is no influence: FlossAR has 122.1 (higher than Ant), OpenBravo 137.5, and OpenERP 67.14 (higher than Blender). OpenERP and OpenBravo are ERP systems (management systems) and there are several large classes (in terms of lines of code) related to database writing and reading operations.

System	Min	1st Qu	Median	Mean	3rd Qu	Max	Sd
Ant	0	12	34	80.87	90	1586	137.52
Blender	1	9	18	38.14	37	1892	80.91
Weka	2	8	35	130.9	147	4078	247.27
FlossAR	2	41	96	122.1	142	6198	204.8
Japs	0	27.75	54.5	78.78	102.2	595	78.82
OpenBravo	1	24	65	137.5	156	3330	229.7
OpenERP	1	12	29	67.14	70	3686	143.15
Zope	1	4	11	33.57	33	1227	72.4

Table 2: Lines of Code

From Tab.3 we considered the number of lines related to code comments. Tab.3 shows that systems developed using Python have a low average value of CLOC (Blender 3.32, OpenERP 4.4, Zope 3.04) and one tentative proposal might be that the design philosophy of Python language emphasizes code readability reducing comments.

System	Min	1st Qu	Median	Mean	3rd Qu	Max	Sd
Ant	0	1	13	45.45	47	1319	94.92
Blender	0	0	0	3.32	2	289	13.16
Weka	0	0	12	65.82	83.75	1141	115.77
FlossAR	0	4	23	36.14	47	1342	57.47
Japs	0	0	5	13.05	18	122	19.55
OpenBravo	0	0	3	21.98	18	870	59.7
OpenERP	0	0	0	4.4	2	384	17.85
Zope	0	0	0	3.04	1	303	11.94

Table 3: Comment line of code

The second result is that systems developed using Agile methodologies are less commented than systems developed using plan-driven methodologies. Support for this interpretation comes from fast development and frequent releases characterizing Agile methodologies. Data from Tab.4 shows that also considering the DIT metric it is not possible to highlight the use of Agile methodologies. As tab.4 shows, there is not a significant difference (Mean column) between the two groups. Differences exist between software developed in Python and software developed in Java.

System	Min	1st Qu	Median	Mean	3rd Qu	Max	Sd
Ant	0	1	2	2.29	3	7	1.23
Blender	0	1	1	1.03	1	6	0.61
Weka	1	1	1	1.89	2	7	1.38
FlossAR	1	1	1	1.71	2	5	0.87
Japs	1	2	3	2.55	3	6	1.08
OpenBravo	0	1	2	2.38	4	5	1.3
OpenERP	0	1	1	1.47	2	5	0.89
Zope	0	1	1	1.66	2	10	1.43

Table 4: DIT

System	Min	1st Qu	Median	Mean	3rd Qu	Max	Sd
Ant	2	17	29	42.37	58	663	34.44
Blender	0	1	2	3.57	3	119	6.78
Weka	13	14	21	95.26	51.75	890	200.8
FlossAR	13	21	25	27.53	30	112	11.92
Japs	13	22	30	36.21	44	114	20.02
OpenBravo	1	18	33	47.01	93	135	32.73
OpenERP	0	1	3	13.33	12	150	26.47
Zope	0	1	4	11.77	12	251	23.64

Table 5: RFC

System	Min	1st Qu	Median	Mean	3rd Qu	Max	Sd
Ant	0	2	4	8.08	10	125	10.82
Blender	0	1	2	2.39	2	84	4.41
Weka	0	1	3	8.63	11	127	12.5
FlossAR	0	4	9	10.85	13	99	10.2
Japs	0	3	6	8.74	12	55	7.72
OpenBravo	0	2	4	7.2	8	122	10.44
OpenERP	0	1	2	3.83	4	106	6.93
Zope	0	0	2	3.42	4	207	6.65

Table 6: WMC

Tabs.5 and 6 show that for the RFC and WMC metrics it is not possible to distinguish the use of Agile methodologies. No significant differences were found between the non-Agile set and Agile set. Also in this case, differences exists between software developed in Python and software developed in Java. Figs. 1, 2, 3, present the CCDF ditributions considering LOC, RFC and WMC. It is apparent from these figures that there are not differences (in terms of CCDF) between software developed using Agile methodologies and software developed using plan-driven methodologies. Considering Fig.6 that contains CCDF distributions of the WMC metric, there is an evident difference between Blender and the others systems. The difference is due to the language; Blender is developed using Python and as can be seen from Tab.6, the mean value of WMC for systems developed in Python (Blender, Zope, OpenERP) is lower than the mean value for systems developed using Java.

4.1 Discussion

According to these results, we can now answer to the research questions:
RQ1: Is it possible to recognize the use of Agile methodologies during the software devolopment process analyzing software metrics?
The answer to this research question is negative. According to Tabs 2, 5, 6, 4 and considering Figs. 1,2,3 the metrics distributions of classes across systems are approximately the same. These empirical results suggest that the use of Agile methodologies and programming practices does not influence the distribution of metrics in the classes.
RQ2: Metrics distributions generated from software developed using Agile methodologies are similar to metrics distributions generated from software developed using plan-driven methodologies?
The answer is yes, as it is possible to see in Figs. 1,2, 3, the CCDF ditributions are the same. Small differences exist in the RFC ditribution of Zope, but considering our data it is not possible to assert that there differences due to use of Agile methodologies.
RQ3: It is possible to assert that metrics distribution generated from Agile methodologies are related to better quality of software?
The answer is negative; metrics distributions are practically the same. In systems developed using Agile methodologies, the LOC distribution does not demonstrate major differences. Considering the average values for each software project under consideration, we obtain similar values. Even in systems developed in Python, which still requires fewer lines of code to express a concept (compared to Java), the LOC distribution is similar to LOC distribution obtained from system developed using Java.
In conclusion, the development methodology does not seem to affect metric distributions.

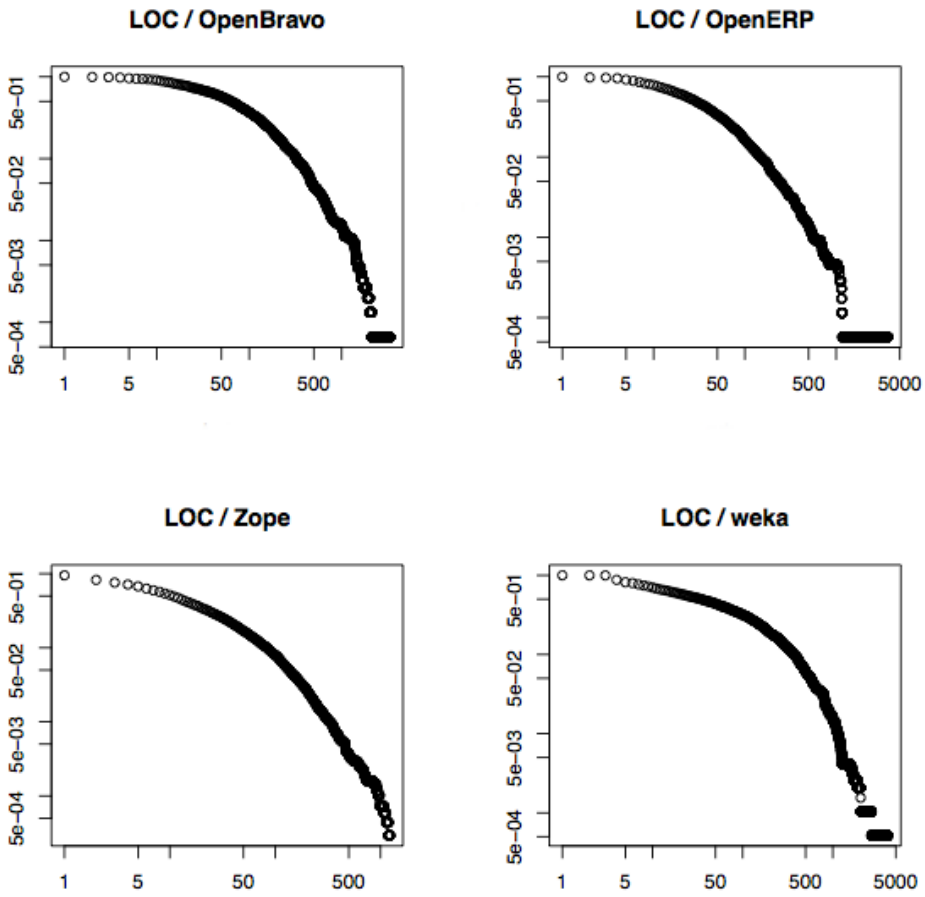

Fig. 1. LOC CCDF distributions

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. 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 Python and this could affect the generality of the study; our results are not meant to 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.

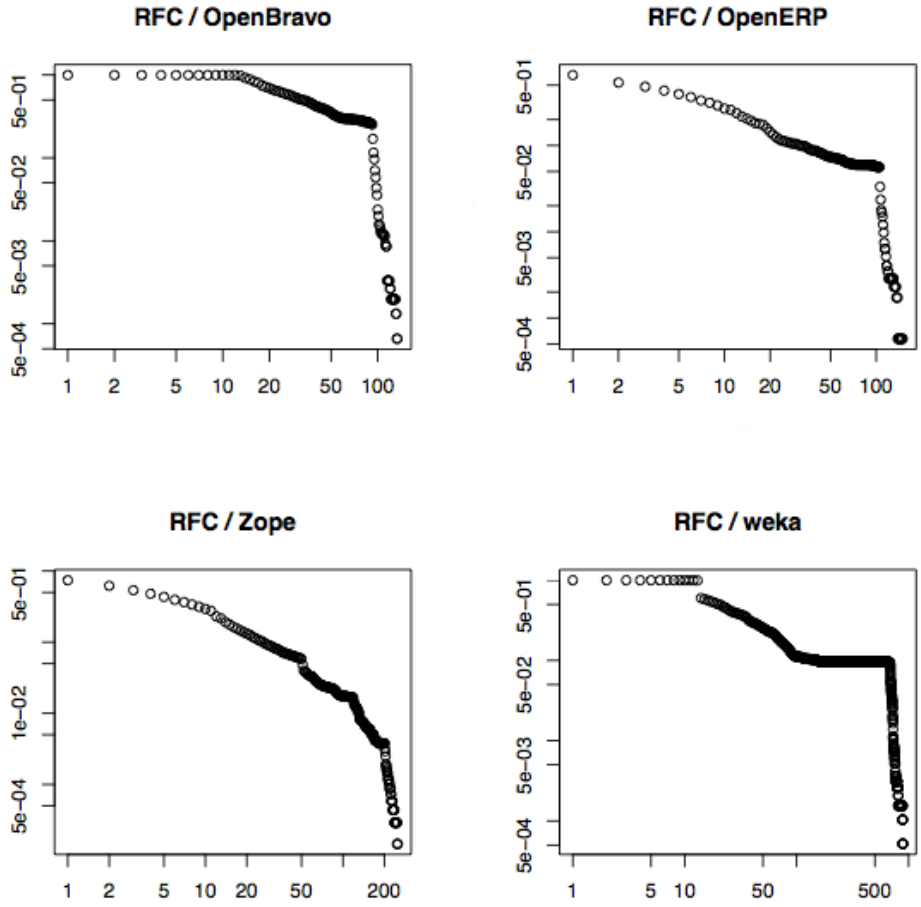

Fig. 2. RFC CCDF distributions

6 Conclusions

In this paper, we presented an analysis of a set of software metrics, performed on 8 number of open source Systems (3 written in Python, 5 written in Java, 5 developed using Agile methodologies, 3 developed using plan-driven methodologies). The size, in classes, of the analyzed projects ranges from 456 to 6852. Overall, we analyzed 17882 classes. The motivation of this work was to understand metric distributions in Agile open source software and to highlight potentially interesting features of software metrics. Our analysis shows that the metrics distribution among systems remains roughly the same as that found in non Agile systems. Thus, the adoption of Agile methodologies does not influence such distribution.

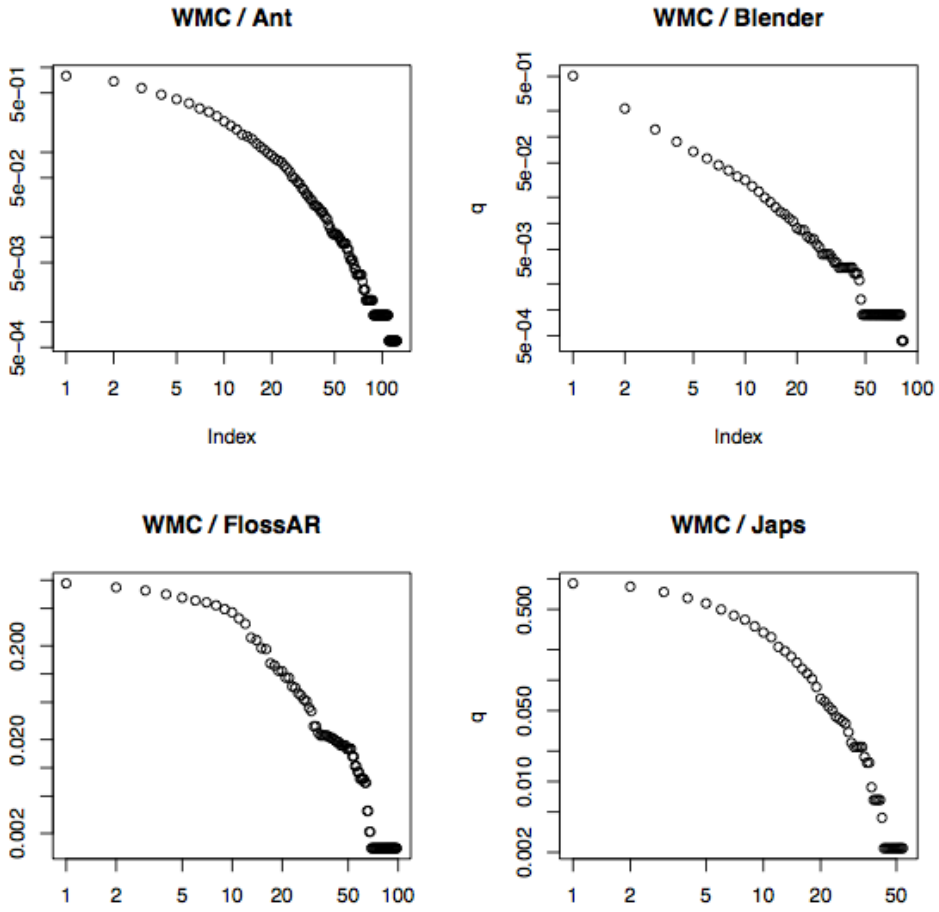

Fig. 3. WMC CCDF distributions

We can conclude that software metrics may be helpful in evaluating 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 metrics. Considering the natural adaptiveness of Agile development, it could be useful to monitor software metrics in order to increase the software quality and decrease the amount of defects.

References

1. Ant: http://ant.apache.org
2. Blender: http://www.blender.org/

3. Weka: http://www.cs.waikato.ac.nz/ml/weka/
4. Floss-AR: http://www.flosslab.it/node/20
5. JAPS: Java agile portal system. URL: http://www.japsportal.org.
6. OpenBravo: http://www.openbravo.com
7. OpenERP: https://www.openerp.com
8. Zope: http://www.zope.org
9. Adams, P. J., and Capiluppi A.: Bridging the gap between agile and free software approaches: The impact of sprinting. Multi-Disciplinary Advancement in Open Source Software and Processes (2011): 54.
10. Adams, P. J., Capiluppi A., and De Groot A.: Detecting agility of open source projects through developer engagement. Open Source Development, Communities and Quality. Springer US, 2008. 333-341.
11. Chidamber S. R., and Kemerer C. F.: A metrics suite for object oriented design. Software Engineering, IEEE Transactions on 20.6 (1994): 476-493.
12. Chidamber S.R., Darcy D., Kemerer C.F., Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis, IEEE Transactions on Software Engineering, v.24 n.8, p.629-639, August 1998
13. Basili V. R., Briand L., Melo W., A Validation of Object-Oriented Design Metrics as Quality Indicators, IEEE Transactions on Software Engineering, v.22 n.10, p.751-761, October 1996
14. Subramanyam R., Krishnan M. S. , Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects, IEEE Transactions on Software Engineering, v.29 n.4, p.297-310, April 2003
15. Concas, G., Marchesi, M., Murgia, A., Pinna, S., & Tonelli, R. (2010, May). Assessing traditional and new metrics for object-oriented systems. In Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics (pp. 24-31). ACM.
16. Concas, G., Marchesi, M., Pinna, S., & Serra, N. (2007). Power-laws in a large object-oriented software system. Software Engineering, IEEE Transactions on, 33(10), 687-708.
17. Brito e Abreu: The MOOD Metrics Set, Proc. ECOOP'95 Workshop on Metrics, 1995.
18. Lorenz M., Kidd J., Object-oriented software metrics: a practical guide, Prentice-Hall, Inc., Upper Saddle River, NJ, 1994
19. Agile Manifesto, URL: www.agilemanifesto.org.
20. Mohammad A., and Wei Li: An empirical study of system design instability metric and design evolution in an agile software process. Journal of Systems and Software 74.3 (2005): 269-274.
21. Alshayeb, Mohammad, and Wei Li: An empirical validation of object-oriented metrics in two different iterative software processes. Software Engineering, IEEE Transactions on 29.11 (2003): 1043-1049.
22. Olague, Hector M., et al: Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. Software Engineering, IEEE Transactions on 33.6 (2007): 402-419.
23. Zhang H.: An investigation of the relationships between lines of code and defects. Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. IEEE, 2009.
24. Dorairaj S., Noble J., and Malik P.: Understanding Team Dynamics in Distributed Agile Software Development, in Proc. XP, 2012, pp.47-61.

25. Bachmann A. and Bernstein A.: Software process data quality and characteristics: a historical view on open and closed source projects. IWPSE-Evol '09 Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, ACM, 2009.
26. Dyb, Tore, and Torgeir Dingsyr: Empirical studies of agile software development: A systematic review. Information and software technology 50.9 (2008): 833-859.
27. Concas G., Marchesi M., Destefanis G., Tonelli R., An empirical study of software metrics for assessing the phases of an agile project, International Journal of Software Engineeringand Knowledge Engineering, Vol 22, 2012, pp.525-548
28. Tasharofi S., Ramsin R.: Process Patterns for Agile Methodologies. In proceeding of: Situational Method Engineering: Fundamentals and Experiences, Proceedings of the IFIP WG 8.1 Working Conference, 12-14 September 2007, Geneva, Switzerland
29. Martin R. 2003. Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR, Upper Saddle River, NJ, USA.
30. Hoda R., Noble J., and Marshall S. 2010. How much is just enough?: some documentation patterns on Agile projects. In Proceedings of the 15th European Conference on Pattern Languages of Programs (EuroPLoP '10). ACM, New York, NY, USA, , Article 13 , 13 pages.
31. Martinez J., Diaz J., Perez J., Garbajosa J.: Software Product Line Engineering Approach for Enhancing Agile Methodologies, Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Business Information Processing Volume 31, 2009, pp 247-248.
32. Olague, Hector M., et al.: An empirical validation of objectoriented class complexity metrics and their ability to predict errorprone classes in highly iterative, or agile, software: a case study." Journal of software maintenance and evolution: Research and practice 20.3 (2008): 171-197.
33. Hartmann D., and Dymond R.: Appropriate agile measurement: Using metrics and diagnostics to deliver business value. Agile Conference, 2006. IEEE, 2006.
34. Frank M., and Martel S.: On the productivity of agile software practices: An industrial case study. Retrieved September 20 (2002): 2004.
35. Concas, G., Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R. (2013): Micro Patterns in Agile Software. In Agile Processes in Software Engineering and Extreme Programming (pp. 210-222). Springer Berlin Heidelberg.
36. Olague, Hector M., et al: An empirical validation of objectoriented class complexity metrics and their ability to predict errorprone classes in highly iterative, or agile, software: a case study. Journal of software maintenance and evolution: Research and practice 20.3 (2008): 171-197.