Open Source, Agile and Reliability Measures

Sharifah Mashita Syed-Mohamad and Tom McBride

University of Technology, Sydney, Australia

Abstract

As open source and agile developments do work in some circumstances, particularly with regard to the release early and often policy, we wonder whether the defect profile (reliability growth) found in the open-source projects so far is typical of open-source software or more generally of software developed using agile approaches. To investigate this, we examined two software products created at a world renowned software organization who focuses on agile software development. The products initially developed for their internal use and after a while were released as open source software. The results of this analysis indicate two findings. First, it supports the tentative findings that agile developed software does not exhibit a standard reliability growth in the defect modeling, and second, somewhat surprisingly that the defect density is reducing, as a sign of improving in quality yet the normal measures of software reliability are not useful.

1 Introduction

Some of the methods with which software quality was judged were based on software development practices are no longer used. Instead of the formal code freeze and stabilization that is common in large formal projects, many of the more agile projects rely on faster feedback either from the user community or from constant testing to gain some idea of the quality of their product. Most agile methods include several practices such as iterative and incremental delivery, test-driven development, self-organizing teams and close customer involvement. An increasing number of organizations are implementing such practices that allow them to be more agile and responsive to the changing market place [Bre 05].

However, one current limitation of agile methods is how their effectiveness is measured, especially when dealing with software quality? Our current study suggests that some of the established measures of software quality, specifically reliability, are not suitable for agile software developments.

Measures are essential in any software development processes, as emphasized by Humphrey, “without measurements, no serious quality program can be effective” [Hum 00; Hum 06]. There are three principal quality measures; time, size and defects. A primary indicator of software quality is defects, highlighted by Humphrey in the Personal Software Process (PSP) or Team Software Process (TSP) [Hum 00].

Software reliability is one of the characteristics of software quality [ISO 06] that is concerned with failures or defects. Usually, when modeling defect data, a defect detection pattern can be observed and an interpolation between data points and an extrapolation of those data points can be performed. Such modelling approach is called software reliability growth [Jel 72; Ohb 84; Goe 85; Mus 87]. In this, the software is usually stabilised through freezing the functionality at the current level, and fixing defects as they are detected during the testing phases. The rate of detection, fixing and overall decline in the number of outstanding defects indicates when the product will reach a level of reliability that it can be released. Using the reliability growth models, developers and customers can know as early as possible the likely quality of their software product.

In our previous work much attention was paid to open source software reliability growth using Orthogonal Defect Classification (ODC) [Sye 08a; Sye 08b]. So far, we found that open source software has a different defect profile to in-house software products possibly due to short release cycles. It seems that the release early and often policy, and the absence of a product stabilisation phase, affect the growth and decline of the defects during development and testing. Consequently, we were not able to fit a meaningful reliability model. Our findings prompt us to ask whether the defect profile found in the open-source projects so far is typical of open-source software development or, more generally, of agile types of development processes. Our ultimate objective here is to characterize the reliability of agile developed software. Answers to this question will help customers to understand software reliability especially for those concerned about trade-offs between reliability and fast delivery of such developments.

In this paper, we have examined two software products. These products were created by a team of programmers and testers at a world renowned software company who focuses on agile software development. After a while, both products were released as open source. We model the entire post-release defects to observe reliability growth patterns of these software products. We also calculate defect density, a *de facto* standard measure of software quality. Findings from this study highlight the need to examine a useful way to measure software reliability in agile types of development methods. This paper proceeds by first reviewing existing work concerning reliability analyses of open source software (see Section 2), and highlighting some issues that we have encountered. We then present an overview of our approach (see Section 3), present our analysis and findings (see Section 4) and provide some discussion and limitations (see Section 5). Finally, we present a summary and highlight our future work (see Section 6).

2 Related work and motivation

In this section, we review some existing work of open source software reliability analyses. We then briefly discuss issues arising from this review.

2.1 Software reliability measures

Software reliability is often defined as the probability of failure-free operation of a software system during a specified time in a specified use environment [Goe 85; Mus 87]. Defect density and reliability growth are essentially two indicators of reliability that help developers in estimating quality of their software system before delivery. To measure defect density, defect counts are normalized by product size, usually measured in lines of code, to gain an overall guide of the code quality.

Reliability growth models on the other hand, model defect data over test time [Lyu 96]. Briefly, reliability growth is the improvement of a software system to deliver proper service. Since defects are introduced in new or changed code, reliability can increase when no new functionalities are added to a software system and defect fixing does not introduce more defects. That is, the rate of defect fixing exceeds the rate of defect introduction or discovery. The two common curves of reliability growth models when plotted over cumulative number of defects against test time are the Concave and S-shaped curves [Lyu 96; Woo 96].

2.2 Reliability growth analyses of open source software

Many studies of open source software development often refer implicitly to the theory of the reliability growth models [Li 05; Tam 05a; Tam 05b; Fen 08]. Open source software reliability growth analyses however, have been inconsistent in their results and conclusions. For instance, [Li 05] described defect detection rates generally increase at the time of release and consequently it is infeasible to fit a meaningful reliability model. Whereas, [Tam 05a; Tam 05b] consider a Logarithmic Poisson execution time model with the effect of debugging process on an entire open source system. They assumed that an open source code has an infinite number of failures due to the effect of the interaction among software components. Examining reliability in distributed environment such open source software leads them to account for the deeply-intertwined factors such as skills of defect reporter and size of each components. Another related research, [Fen 08] finds that the traditional reliability growth models are generally not suitable for assessing the reliability of open source software. Unevenly distributed defect detection among releases is the reason for the unsuitability. It is infeasible, for example, to fit a reliability model to only three data points for a release.

Given these inconsistent results in repeatability growth analyses of open source software, [Sye 08a] made another attempt to investigate this by examining reliability growth using the Orthogonal Defect Classification (ODC). Their initial findings based on two open source projects show that open source software has a different defect profile to in-

house software, where no stabilization curves are observed. Hence, open source software has a different reliability growth to in-house software. Fig. 1 shows the reliability growth of open source and in-house software in relation to the ODC. The ODC work reveals that open source software appears to be unstable in the area of low level design i.e. 'Interface', 'Serialization' and 'Algorithm' types of defect. Moreover, the authors deduce that short release cycles as the cause of the difference where this affects growth and decline of defects. Taken together, these findings motivate us to investigate whether the defect profile found in the open-source projects so far is typical of open-source software development or more generally of agile developed software.

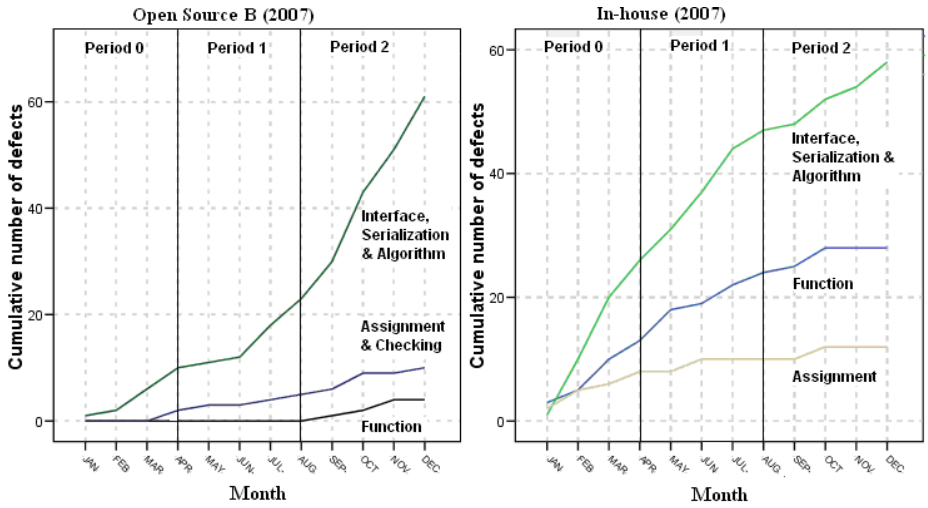

Fig. 1: Reliability growth curves for the collapsing of categories based on ODC defect type [Sye 08a].

2.3 Rapid change and short development cycles in agile

Agile is often described as light-weight processes in contrast to plan-based processes in the waterfall approach. Agile methods embrace change in a fast, iterative and incremental manner. They have short development cycles, close customer participation and do deliver working software at each cycle. Extreme Programming [Bec 99] and Scrum [Bee 02] are the two prominent agile approaches.

Given that agile does work in some circumstances such as open source software development, particularly with regard to the shorter and frequent release policy, we want to discover reliability growth in agile software development. In this paper, we will examine post-release defect data to see whether or not agile developed software exhibits a standard reliability growth in the defect modeling.

3 Approach

In this section, we provide information about the application software under examination and elaborate our approach to obtain the defect profile of these applications.

3.1 Software product descriptions and data collection

Two software products namely Agile A and Agile B (not their real names) have been chosen in this case study. Both products were initially created by a team of programmers and testers at a world renowned software organization who focuses on agile software development for their internal use. After a while, the products have been released as open source. Agile A is a development tool for browser-based testing of web applications. It can be used both for functional and compatibility testing. Most of this product is written in JavaScript, with additional files written in XML, HTML and CSS. We examined eleven releases of Agile A which span over more than 4 years of development. The first release was 0.2 and the last release under investigation was 0.8.3. Tab. 1 lists the releases information in details.

Agile B is a continuous integration tool that is extensible for creating a custom build process. We examined fifteen releases of Agile B; as listed in Tab. 2. Most of this software is written in Java, with a number of additional files written in JavaScript, XML, XSLT, HTML, JSP and CSS to provide a web interface to view details of the current and previous builds. As many other open source products, both Agile A and B do not have fixed release schedules.

We gathered defect data from Jira, an Atlassian's issue tracking system used in tracking any issues of Agile A and B. These issues can be defects, feature requests, improvements or any other tasks the developers want to track. From our observation of the defect dataset we

Tab. 1: Release plans and accepted number of defects for Agile A

#	Release Version	Release Date	Post-release defect
1	0.2	20-Jan-05	15
2	0.3	2-May-05	14
3	0.4	20-May-05	5
4	0.5	19-Jun-05	4
5	0.6	24-Sep-05	39
6	0.7.0	14-May-06	24
7	0.7.1	3-Aug-06	25

8	0.8.0	20-Sep-06	35
9	0.8.1	13-Nov-06	26
10	0.8.2	11-Dec-06	38
11	0.8.3	20-Sep-07	34

Tab. 2: Release plans and accepted number of defects for Agile B

#	Release Version	Release Date	Post-release defect
1	2.1.5	05.02.2004	19
2	2.1.6	30.06.2004	17
3	2.2	29.10.2004	12
4	2.2.1	01.02.2005	59
5	2.3	28.08.2005	15
6	2.3.1	29.09.2005	33
7	2.4	20.01.2006	2
8	2.4.1	28.02.2006	18
9	2.5	24.04.2006	62
10	2.6	12.01.2007	19
11	2.6.1	28.02.2007	20
12	2.6.2	22.04.2007	7
13	2.7	02.06.2007	19
14	2.7.1	29.08.2007	43
15	2.7.2	02.04.2008	27

find that the developers consistently and persistently supervise and maintain their issue tracking system. They clearly classified each issue reported to their issue tracker either as a defect or improvement or new feature or task. Such categorizations enable us to quickly perform data separation. We left out improvement, new feature and task issues in order to consider only defects in our study. Knowing that data filtering is essential to obtain an accurate reliability analysis as suggested in previous studies [Kan 97], so the next task was to check defects that should not be considered in this study. We removed

'duplicate', 'invalid', 'not a problem' and 'not reproducible' resolutions from our consideration. Also, we excluded specification and background defects such as installation or platform problems, to be consistent with our previous approach [Sye 08a]. Overall, we find that the quality of the defect reports was good. Our final task was to count and group the accepted defects into particular release versions. Most of the defects have affect version information and those that did not have one; were classified according to the release date. For instance, a defect that occurs after one release date was considered belongs to the release version. This approach can also be found in [Li 05].

4 Results and observations

We now describe the reliability growth results of Agile A and B. In addition to that, we present our observation on using defect density to measure the overall quality of these agile developed software.

4.1 Reliability growth analyses

Agile practices emphasize using test automation and tools. By automating their testing, agile teams can run the same tests over and over again. This helps them to ensure their incrementally added codes to the software system do not break anything that the previous one used to work and do what it is supposed to do. For this reason, we wonder about the amount of defects in case such development practices produce fewer defects and we may not be able to do the defect modeling. In both products we found considerable number of defects for each release. This enables us to monitor the growth and decline of the defects, so reliability growth analyses can be performed.

First, we modeled the overall reliability growth by plotting the cumulative number of post-release defects over the entire development period. This means that we examined the stabilization curve of the products over multiple releases. Fig. 2 and Fig. 3 illustrate the overall reliability growth curves of Agile A and B, respectively. Obviously, both curves do not show a standard reliability growth. The defect rate of Agile A stabilizes for the first 15 months but then rapidly increases until it declines again at around 24 months in the development. Interestingly, from the figure, it is evident that the defect arrival rate of Agile A increases rapidly after many releases are delivered in short period of time. As for Agile B, the curve is linearly increasing over time with no sign of stabilization. These results support our previous study that open source and agile do not exhibit the same reliability growth pattern as in the traditional reliability growth model, where a stabilization curve should be observed.

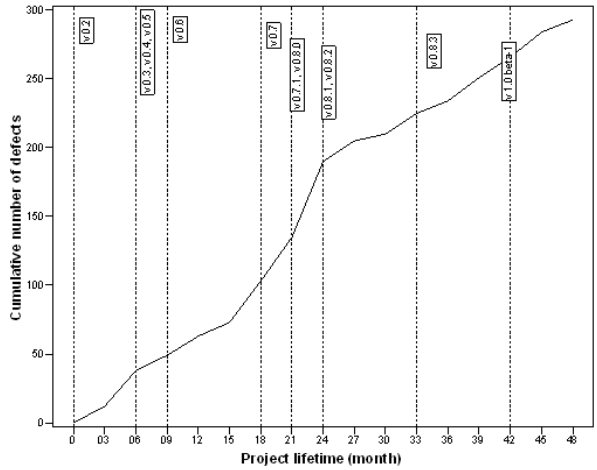

Fig. 2: Defect modeling of Agile A.

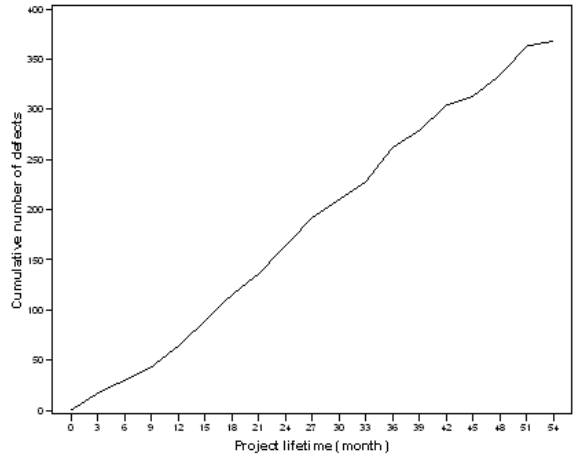

Fig. 3: Defect modeling of Agile B.

Then, we made an effort to model post-release defects for particular releases of Agile A and B to measure reliability growth of individual releases. We employed concave, S-shaped and inflection S-shaped models (the same models that we used in previous work) to fit the defect modeling. Results of these models are shown in Fig.4 and Fig.5. We did not obtain a good fit model (R^) of either concave or S-shaped or inflection S-shaped for many releases. Briefly, R squared as an indication of how good the correlation between the cumulative number of defects to the time after release. R^ of concave, S-shaped and inflection models for release 0.6 of Agile A are 0.783, 0.915 and 0.661, respectively. Even poor results were obtained for release 2.7 of Agile B, as shown in Fig.5. Possibly, all of this tells us that prediction of future reliability or defect-prone re-

leases based on defect modeling can not be identified early enough in agile software development. Defect prone analyses previously have helped developers to control and manage quality of future releases [Chi 95; Kho 00].

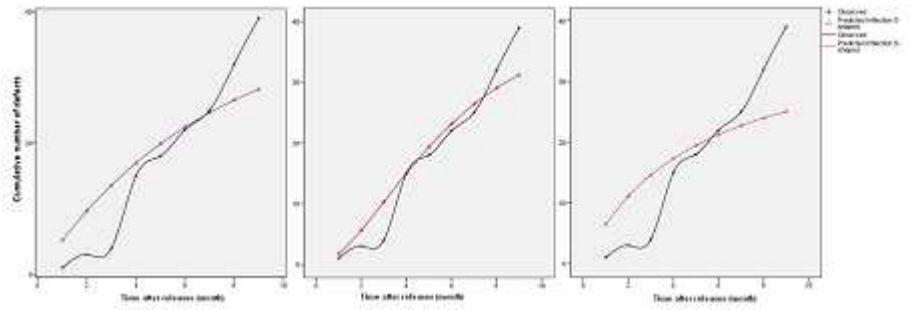

Fig. 4: Concave (left), S-shaped (middle) and Inflection S-shaped (right) growth modeling of Agile A version 0.6.

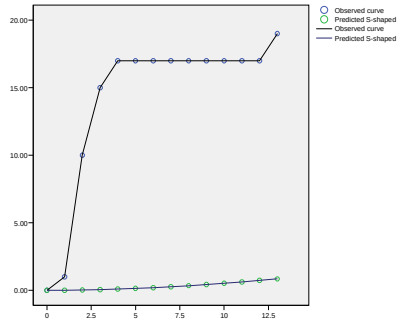

Fig. 5: S-shaped growth modeling of Agile B version 2.7.

4.2 Defect density measures

As we were not able to measure reliability from the defect modeling, we then measure defect density to gain some idea of the quality of Agile A and B. Defect density has been commonly used in many prior quality studies [Mal 90; Kit 96; Hum 00]. This quality measure refers to the number of known defects per product size, where product size is usually measured in lines of code (LOC). Normalizing defect counts by product size allows us to compare the quality of products that differ greatly in size and helps determine the effectiveness of the development activities [Kit 96], such as defects finding activities.

We examined source codes from the last six releases of Agile A and all releases of Agile B. We employed CLOC (http://cloc.sourceforge.net/) to calculate the product size. The product size is measured in physical uncommented source lines. The current size of Agile A is about 20,000 LOC. We removed from consideration the product index, refer-

ence and non-code files like word files, gif, jpg and readme files. The detailed results are listed in Tab.3. Note that we were not able to examine defect density of version 0.2 until 0.6, due to the source codes unavailability.

From Tab.3 several points are worth noting. First, there is just as likely to be a defect introduced in a defect fix as there is with a new feature or improvement task. Second, the quality of Agile A is improving as the defect density is reducing from 2.88, reaching 1.58 defects/KLOC and fluctuating around the average, as depicted in Fig. 6. We can expect that the defect density to be better in future releases. Third, we should notice that the average defect density is 2.20 defects/KLOC in which it represents the delivered quality to the users. Most software engineers consider a software product with delivered defect density of below 2 per KLOC to be very good [Nor 97], so, the overall quality of Agile A is considered good.

As for Agile B, its size is growing from around 22,000 to almost 140,000 non-comment lines of code. The release 2.7 was released with significant addition in features that allows users to help visualizing the project build statuses with colour coded in previous project build result. The overall quality of this product is also good since the delivered defect density of each release is around 0.24 to 3.51 defects/KLOC. As can be seen in Fig. 7 (due to space limitation no table is presented) defect density of the last several releases is fluctuating just around 0.2 to 1 defects/KLOC. From all of this we assume that the overall development process is improving the software product particularly, their defect finding techniques. We briefly explain our assumptions in the discussion section.

Tab. 3: Defect density results

Release version	**Development day**	**Post-release defect**	**Fixed defect**	**Improvement, task & New Feature**	**Lines of code**	**Defect Density (Defects/KLOC)**
0.7.0	232	24	17	17	8326	2.88
0.7.1	81	25	6	5	11212	2.23
0.8.0	48	35	13	11	15080	2.32
0.8.1	54	26	16	8	16475	1.58
0.8.2	29	38	20	5	16820	2.26
0.8.3	283	34	13	12	17412	1.95

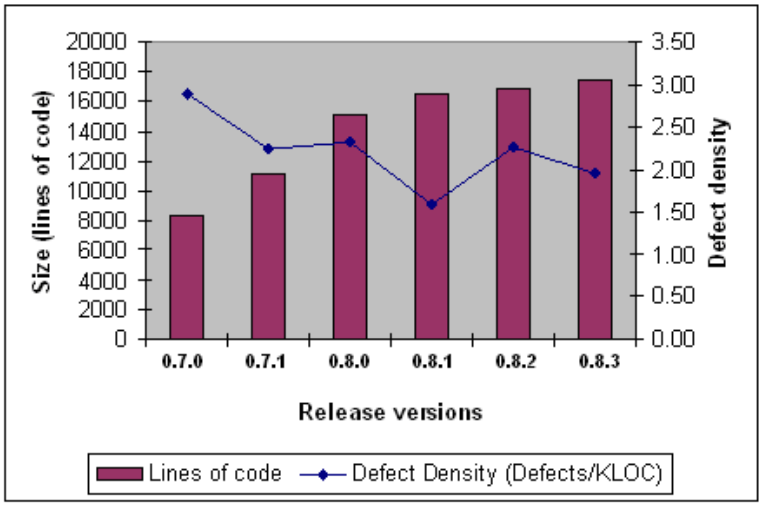

Fig. 6: Defect density vs. product size for Agile A.

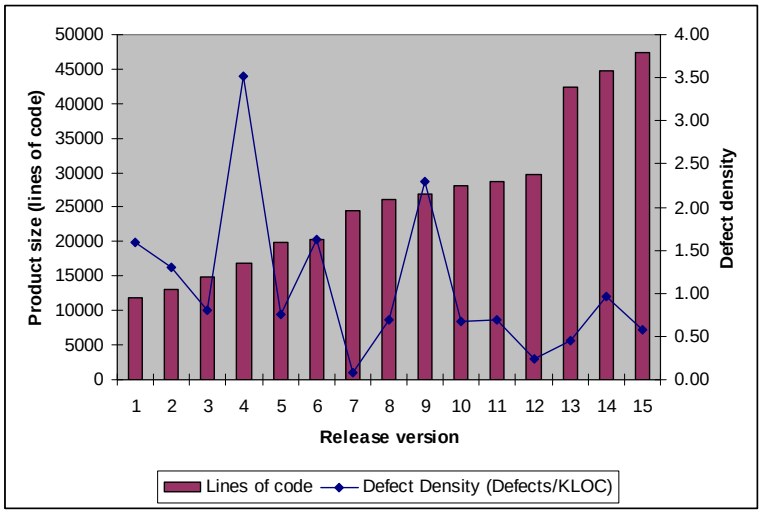

Fig. 7: Defect density vs. product size for Agile B.

In summary, both of the agile developed software differs with respect to product size, programming languages, number of defects, and number of releases. The difference and similarities of these products are shown in Tab. 4 and Tab. 5. We confirm some general statements such as the overall defect modeling over several releases and defect density. The variations between the products are considered important factors in increasing the understanding of defect distributions.

Tab. 4: The differences between the software products

Criteria/Software product	Agile A	Agile B
Product size (of the latest release)	61884	139101
Programming language (main)	JavaScript	Java
Number of releases examined	11	15
Number of defects	259	372

Tab. 5: The similarities of Agile A and B

Criteria/Software product	Agile A and Agile B
Type of development	Agile developed software
Number of developers	Between 50 and 100
Product domain	Software development

5 Discussion and limitation

From our previous work, we have tentatively established that open source software does not exhibit a standard reliability growth in the defect modeling. This brings us to find out whether the defect profile found in the open source projects is typical of open source or of software developed using agile processes. In this paper, we examine stabilization curve over several releases of two agile developed software. The result supports our previous findings as we observed no stabilization curves in the defect modeling. It seems that most of the software reliability models have been developed in the traditional 'big bang' area where the software is largely stabilized before testing and release, hence the concept that defects will be discovered in a characteristic profile.

But this is not the case for software that is continually being expanded and modified. Agile and open source software involve with frequent deliveries of new and corrected functionality. As described in [Lyu 96], if the software is being tested or observed changes considerably from the one in which the data have been collected, perfect prediction of future behaviour can not be expected. "The software must have matured to the point that extensive changes are not being routinely made". For this reason, the traditional reliability growth models might not really suit for agile types of development methods. Thus, we face some new challenges in modelling software reliability for agile

developed software. What we most concern is the possibility of such development to build high reliability software as possible in short time periods.

However, results from defect density indicate somewhat surprisingly that the defect density, a *de facto* standard measure of software quality, is improving over several releases despite increasing in code size yet the normal measures of software reliability are not useful. Improving in the defect density may tell us about the effectiveness of defect finding process. One reason that we can assume of the improving in quality is the efficiency of the overall development process, where in this case, agile methodologies. It seems that their testing, particularly, automated regression testing results in reducing delivered defect density, despite in adding functionalities and fixing defects. Regression testing by nature can help developers to ensure that their added code to the system does not break anything that the previous one used to work and it does what it is supposed to do. From our examination on the product test suite, the developers employ automatic unit testing. As for Agile A, the automatic test scripts written in HTML & JavaScript and jUnit test framework for Agile B. Developers of both products persistently maintained their test code. The test suite seems to grow uniformly with the production code. This supports our assumption that the improvement is likely due to the techniques.

However, this paper reports a study conducted on only 2 projects. We assume the products are representative of one type of development method that is agile since the main developer and maintainer of this product is a software organization that focuses on agile development. More importantly, the main point that concerns us here is the reliability growth measures may not be expected to give good estimations of reliability in agile software developments. Hence, other reliability measures should be studied. We need to examine more projects that cover wide range of application domains, programming languages, product sizes, number of developers and types of software development before reaching any firms conclusions.

6 Conclusion and future work

The traditional reliability growth models have been used as a guide help with measure and achieve reliability resulting from changes and fixes made during the development, testing and stabilization periods. Basically, for software without new functionalities are routinely added and new defects are introduced during corrective actions, stabilization curve can be identified.

In this paper, we examined two software products created by a software organization to investigate reliability growth patterns in agile types of development processes. The results of this study indicate two findings. First, it supports the tentative findings that agile developed software does not exhibit a standard reliability growth in the defect modeling, and second, somewhat surprisingly the examined software products exhibit a decline in the defect density over several releases despite increasing in code size and this tells us that something in the software development process is improving the software. As these developments used automated regression testing, we deduce that their defect

finding activities are somewhat efficient which result in reducing delivered defect density despite in adding functionalities and fixing defects.

We now intend to examine more software projects that represent agile types of development method, and find a useful way to measure software reliability of agile developed software, in order to understand the quality of a software product/system.

References

[Bec 99] Beck, K. : Extemme Programming explained. Embrace change, Addison-Wesley. 1999

[Bee 02] Beedle, M., Devos, M., Sharon, Y., Schwaber, K. and Sutherland, J. : 'SCRUM: An extension pattern language for hyperproductive software development'. 2002

[Bre 05] Brechner, E. : 'Journey of enlightenment: the evolution of development at Microsoft', Software En gineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 39-42. 2005

[Chi 95] Chillarege, R., Biyani, S. and Rosenthal, J. : 'Measurement of failure rate in widely distributed soft ware', Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on, pp. 424-433. 1995

[Fen 08] Fengzhong, Z. and Davis, J. :'Analyzing and Modeling Open Source Software Bug Report Data', Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on, pp. 461-469. 2008

[Goe 85] Goel, A. L. :'Software Reliability Models: Assumptions, Limitations, and Applicability', Software Engineering, IEEE Transactions on, Vol.SE-11**,** no no.12**,** pp. 1411-1423. 1985

[Hum 00] Humphrey, W. S.:The personal software process (PSP), Technical Report CMU/SEI-2000-TR-022 Carnegie Mellon University. 2000

[Hum 06] Humphrey, W. S.: Systems of Systems: Scaling up the development process, Technical Report CMU/SEI-2006-TR-017 Carnegie Mellon University. 2006

[ISO 06] ISO/IEC WD 25010 : Software engineering - Software product Quality Requirements and Evalua tion (SQuaRE) - Quality model . 2006

[Jel 72] Jelinski, Z. and Moranda, P. B. :'Software reliability research', Statistical Computer Performance Evaluation**,** pp. 465 - 484. 1972

[Kan 97] Kanoun, K., Kaniche, M. and Laprie, J.-C.:'Qualitative and Quantitative Reliability Assessment', IEEE Softw., Vol.14**,** no 2**,** pp. 77-87. 1997

[Kho 00] Khoshgoftaar, T. M., Allen, E. B., Jones, W. D. and Hudepohl, J. P.:'Accuracy of software quality models over multiple releases', Annals of Software Engineering, Vol.9**,** no 1-4**,** pp. 103-116. 2000

[Kit 96] Kitchenham, B. and Pfleeger, S. L.:'Software Quality: The Elusive Target', IEEE Softw., Vol.13**,** no 1**,** pp. 12-21. 1996

[Li 05] Li, P. L., Herbsleb, J. and Shaw, M. : 'Finding predictors of field defects for open source software sys tems in commonly available data sources: a case study of OpenBSD', Software Metrics, 2005. 11th IEEE International Symposium, pp. 10 pp. 2005

[Lyu 96] Lyu, M. R.: Handbook of Software Reliability Engineering, Michael, R. L. (Ed), McGraw-Hill, Inc.
1996

[Mal 90] Malaiya, Y. K. and Srimani, P. K.: Software Reliability Models: Theoretical Developments, Evalua tion, and Applications, IEEE Computer Society Press. 1990

[Mus 87] Musa, J. D., Iannino, A. and Okumoto, K.:Software reliability: measurement, prediction, application. 1987

[Nor 97] Norman, E. F. and Pfleeger, S. L.: Software metrics : a rigorous and practical approach . PWS Pub. 1997

[Ohb 84] Ohba, M.: 'Software reliability analysis models', IBM Journal of Research and Development, Vol.28**,** no 4**,** pp. 428-443. 1984

[Sye 08a] Syed-Mohamad, S. M. and McBride, T.: 'A Comparison of the reliability growth of open source and in-house software', 15th Asia-Pacific Software Engineering Conference (APSEC 2008), Beijing,
China, IEEE Computer Society, pp. 229-236. 2008

[Sye 08b] Syed-Mohamad, S. M. and McBride, T.: 'Reliability growth of open source software using defect analysis', International conference on computer science and software engineering, Wuhan, China, IEEE Computer Society, pp. 1-6. 2008

[Tam 05a] Tamura, Y. and Yamada, S.: 'Comparison of software reliability assessment methods for open source software', Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference
on, pp. 488-492 Vol. 2. 2005

[Tam 05b] Tamura, Y., Yamada, S. and Kimura, M.: 'Reliability Assessment Method based on Logarithmic Poisson Execution Time Model for Open Source Project', Proceeding (489) ACIT - Software Engineering. 2005

[Woo 96] Wood, A.:Software Reliability Growth Models, Tandem Tech. Report 96-1, Tandem Computers. 1996