

Predicting Software Defect Density: A Case Study on Automated Static Code Analysis

Artem Marchenko¹ and Pekka Abrahamsson²

¹ Nokia, Hatanpäänkatu 1, FIN-33100 Tampere, Finland

² VTT Technical Research Centre of Finland,

P.O.Box 1100, FIN-90571 Oulu, Finland

artem.marchenko@nokia.com, Pekka.Abrahamsson@vtt.fi

Abstract. The number of defects is an important indicator of software quality. Agile software development methods put an explicit requirement on automation and permanently low defect rates. Code analysis tools are seen as a prominent way to facilitate the defect prediction. There are only few studies addressing the feasibility of predicting a defect rate with the help of static code analysis tools in the area of embedded software. This study addresses the usefulness of two selected tools in the Symbian C++ environment. Five projects and 137 KLOC of the source code have been processed and compared to the actual defect rate. As a result a strong positive correlation with one of the tools was found. It confirms the usefulness of a static code analysis tool as a way for estimating the amount of defects left in the product.

Keywords: agile software development, static code analysis, automation, defect estimation, quality, embedded software, case study.

1 Introduction

The number of defects has generally been considered an important indicator of software quality. It is well known that we cannot go back and add quality. By the time you figure out you have a quality problem it is probably too late to fix it. [1]

The embedded software industry faces a number of the specific quality related challenges. The embedded devices software typically cannot be updated by the end user. In the majority of cases the software problems can be fixed only at the authorized maintenance centers. The devices running the embedded software have both hardware and software based components. Nokia and other mobile terminal manufacturers release dozens of mobile phone types a year. It significantly scales the amount of the required maintenance effort and number of software configurations to be supported.

Agile software development teams use automated tools to constantly be aware of the quality of a running system. One of the sources of the metrics analyzed is the static code analysis tools. While these tools are not able to spot all possible defect types, their reports may correlate with the actual number of significant defects in software. If such correlation is found, it will make the static code analysis an important element of the agile team toolbox for getting the quality related view on the

developed code. Currently the main drawback of the static code analysis is the lack of empirical evidence of the correlation between the tool reports and the actual defects rate. There is also no explicit evidence in the area of embedded software that the use of automated static source code analysis would yield results that confirm the correlation between the actual defect rate and predicted defect rate.

This paper presents a case study on predicting defects in the domain of embedded software development by use of automated static code analysis tools. The suitability of two particular tools, i.e. CodeScanner and PC-LINT, is tested on a number of components shipped as a part of Nokia smartphone software. The feasibility of a broader study is indicated.

2 Related Literature

Fenton and Neil (1999) outline four general approaches to predicting the number of defects in the system. [2]. This article is based on the approach of finding the correlation between the defect density and the code metrics. The metrics used for the defect rate prediction are produced by the process of static code analysis – the analysis of software statically, without attempting to execute it [3].

There are some studies on the static code analysis effectiveness reporting somewhat controversial results. Dromey (1996) suggests that code analysis can find most of quality defects and report them in a way convenient for the developers to correct the code. [4] Nachiappan and Thomas (2005) found that there was a strong correlation between the number of defects predicted by static analysis tools and the number of defects found by testing [5]. On the other hand Lauesen and Younessi (1998) state that the code analysis can locate only a small percentage of meaningful defects [6].

As shown above, currently, there are virtually no studies on applicability of static code analysis tools in the area of embedded software development as is the case in this study, i.e. Symbian operating system environment. This study focuses on evaluating the ability of the static code analysis tools to predict the number of defects in the software written in C++ for the Symbian operating system.

3 Research Design

In this study a number of components of the mobile phone software have been analyzed. All the components are written in C++ for the Nokia S60 software platform based on the Symbian operating system. The source code has been processed by two static code analysis tools: CodeScanner [7] and PC-Lint [8].

CodeScanner is a tool specifically developed for the Symbian C++ code analysis, while PC-LINT is a general C++ code analysis tool that can be fitted to the particular environment. In this case two sets of in-house built Symbian specific rules have been used to fit PC-LINT to the Symbian code idioms (“normal” and “strict” rule sets).

For the issues reported by the tools the “issue rate” per non-comment KLOC has been computed. The actual defect rate has been obtained from the company defect database. The defects reported within 3 and 6 months after the release date have been taken into account.

The projects have been ranked in the order of the rates. The correlation between the ranks has been computed in order to find out if there is a link between the issue rates and the actual defect rates. Spearman rank correlation has been used for the results analysis, because it can be applied even when the association between elements is non-linear. If rank correlation between the issue rate and the defect rate is positive, then for the projects analyzed, the bigger the issue rate is, the bigger defect rate should be expected.

4 Empirical Results and Discussion

The case study included five components of the 3rd edition of the Nokia S60 platform for smartphones. After the testing and debugging related code exclusion, the size of the code analyzed was 137 KLOC.

Table 1 shows the correlations between the reported issue rates and acknowledged defect rates. The first column presents the CodeScanner results. The next three columns contain PC-LINT results with different variants. The first line presents correlation with critical defects that were reported within 90 days and the second line - with the critical defects reported within 180 days after the release.

Table 1. Correlation results of defects/KLOC

Actual defect rate	Code Scanner rate	PC-LINT strict errors rate	PC-LINT strict errors + warnings rate	PC-LINT normal errors + warnings rate
90 days rate	0.7	-0.7	-0.9	-0.7
180 days rate	0.9	-0.6	-0.7	-0.9

For the projects analyzed there is a strong positive correlation between the CodeScanner defect rate and the actual reported defect rate, i.e. 0.7 in 90 days rate and 0.9 in 180 date rate. Interestingly, there is a strong negative correlation between the PC-LINT defect rate and the actual reported defect rate.

A strong positive correlation between the actual defect rate and CodeScanner reported issues confirms the Nachiappan and Thomas (2003) findings that there is a strong correlation between the static analysis defect density and the pre-release defect density reported by testers of the Windows Server 2003 [5].

A strong negative correlation between the PC-LINT reported issues and the actual defect rate might be a result of the unsuccessful attempt to fit the PC-LINT tool to the Symbian specific issues therefore being a confirmation of the Lausen and Younessi (1998) claim that static analysis tools are able to locate only a small number of meaningful defects [6]. Typical Symbian C++ code significantly differs from the typical Win32 or *nix code. Therefore, it might be so that the closer developers adhere to the industry recommended Symbian idioms, the more issues are reported by PC-LINT.

The CodeScanner tool analyzed in this study has been developed specifically for the Symbian OS C++ code and cannot be applied for other embedded software types.

Therefore the study results are less significant outside the Symbian OS area. For two of the projects analyzed the difference between the corresponding CodeScanner issue rates was less, than 1%. It is unclear how reliable the Spearman rank correlation is in such a situation.

It is also not very clear if the tools examined can be used to predict the defect density of a random sample. A larger case study is needed to address these issues.

5 Conclusions

This study aims at contributing to the problem of estimating the software maintenance costs and of evaluating the software quality. The angle of analysis was the ability for using the static code analysis tools for the software defect rate prediction in the area of embedded software development.

The results indicate that static code analysis tools can be used for helping the agile teams to perform better. If broader studies confirm this paper results, agile teams in the domain of embedded software development will get an important tool for quickly and regularly getting the view on the quality state of the software under development. Future research can be aimed at figuring out which issues detected by the tools correlate with the actual defect rate and which do not.

References

1. Reel, J.S.: Critical success factors in software projects. *Software*, IEEE 16(3), 18–23 (1999)
2. Fenton, N.E., Neil, M.: A critique of software defect prediction models. *Software Engineering*, IEEE Transactions on, 1999 25(5), 675–689 (1999)
3. Chess, B., McGraw, G.: Static analysis for security. *Security & Privacy Magazine*, IEEE 2(6), 76–79 (2004)
4. Dromey, R.G.: Concerning the Chimera [software quality]. *Software*, IEEE 13(1), 33–43 (1996)
5. Nachiappan, N., Thomas, B.: Static analysis tools as early indicators of pre-release defect density. In: *Proceedings of the 27th international conference on Software engineering*. St. Louis, MO, USA (2005)
6. Lauesen, S., Younessi, H.: Is software quality visible in the code. *Software*, IEEE 15(4), 69–73 (1998)
7. Newsletter, S.O.C. Symbian OS Community Newsletter - October 19th, 2004 (2004)[cited 2004 24 January] Available from: <http://developer.symbian.com/main/getstarted/newsletterarchive/newsletter31.jsp>
8. Donner, I.: Computer-Related Inventions: When 'Obvious' is Not So Obvious. *Computer* 28(2), 78–79 (1995)