Maintainability of Source Code and its Connection to Version Control History Metrics

Csaba Faragó

Department of Software Engineering
University of Szeged

Szeged, 2016

Supervisor:
Dr. Rudolf Ferenc

Summary of the Ph.D. Thesis

University of Szeged
PhD School in Computer Science

Introduction

If a program would be written exclusively for the computers, and it would be 100% sure that no human will ever see the source code later, then the code quality would not be important. In that case also this thesis would deal with some completely different topic. However, this is not true: the developer spends most of her/his work time on *reading* source code, and not *writing* it. Therefore, the developers should pay attention to the maintainability of the source code to help the subsequent readers, who might be indeed the same developer after a longer while.

Software code quality is very important, because a too complex, hard-to-maintain code results in more bugs on one hand, and makes the further development more expensive on the other hand. Developers typically focus on the problem: to solve it, make it work, and the importance of the code quality in time pressure is frequently secondary. The motivation of the studies behind this thesis is to help developers to create a source code which is easier to maintain.

The thesis consists of two main topics: *program slicing* and the *impact of version control history metrics on maintainability*. There is a strong connection between them: the *maintainability of program source code*.

One of the usages of **program slicing** is to improve the maintenance of the source code, helping the programmer by highlighting the relevant parts of it. With this technique the developer can eliminate the source code irrelevant from a certain problem viewpoint, and observe only those statements which really influence the erroneous part. In this thesis we focus on the unstructured statements handling in a certain dynamic program slicing algorithm.

In the topic of **version control history analysis** we try to find evidence why code maintainability decreases. The seventh law of Lehman is about declining quality, and it states that the quality of a real-world software system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes In the second part of this thesis we deal with the connection between this code decay and some version control history metrics. Although both the topic of maintainability and the software repositories mining are thoroughly analyzed research fields, we are not aware of any study which deals with the, strictly speaking, connection between version control history metrics and the maintainability of the source code. So in this thesis we present a pioneer work in this young research field.

The thesis consists of three thesis points. In this booklet we summarize the results of each thesis point in a separate section.

Thesis Point 1: Unstructured C Statements Handling in a Dynamic Slicing Algorithm

A slice consists of all statements and predicates that might affect a set of variables at a program point. Slicing algorithms can be classified according to whether they only use statically available information or dynamic information as well, to static and dynamic program slicing. In this thesis we deal only with dynamic program slicing.

Gyimóthy et al. [15] introduced a method for the forward computation of dynamic slices. The point of the methodology in a nutshell is the following. We determine statically for every line of code which variable gets value, and on which variables it depends on. We handle the conditional and cycle statements as virtual predicate variables. We instrument the program and execute it to gain the execution history. For every executed step, computing forward we calculate from which lines the actual line depends on. This is the union of the last modification places and their dependent lines of those variables which the actually calculated variable depends on.

In practice we can consider the memory requirement of the methodology to be linear to the memory requirement of the original program, which was a significant improvement compared to the big memory requirements of earlier methodologies. However, the algorithm in its original format was inappropriate for slicing real programs, because it practically handled assignment, conditional and loop statements only.

In their study Beszédes et al. [3] adopted the algorithm on the C programming language. They solved several issues, e.g. the function calls or the pointer handling.

One of the issues to be solved was the handling on unstructured statements in the C programming language, which are the following: `goto`, `break`, `continue` and `switch-case-default`. In our solution [8, 9] we introduced so-called label variables, which get value at the point of execution, and the dependent lines of the source are those located after the label. In the case of `goto` the dependent statements are all the statements after the declaration of the label within the function. In the case of `break` the dependent statements are all the statements after the related code block (e.g. after the `while` block). In the case of `continue` the dependency should be introduced from the first statement of the related code block until the end of the function. This also means that the `continue` always depends on itself. In the case of `switch-case-default`, the `switch` statement should be handled with a predicate variable, similarly to e.g. in the case of the `while` statement. If at least one internal statement is part of the result, then all the `case` labels, along with the `default`, should be put into the result.

Later we extended the method [2], where we defined the relevant slice as the union of the all possible executions. In case of significant code coverage the size of the resulting slice was a fraction of the result calculated by a static program slicing tool.

The author's contributions to the results

The author contributed to the new results presented in this section as follows:

- Elaboration of the unstructured statements handling in the presented dynamic slicing algorithm.
- Participating in the implementation.
- Execution of the tests.

A Figure 1 provides an overview of how these parts fit together, and where the author's contribution is located in the system.

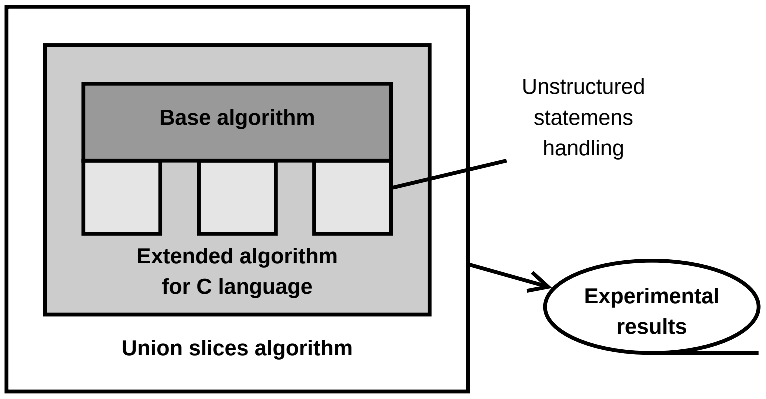
Base algorithm
Unstructured statemens handling
Extended algorithm for C language
Experimental results
Union slices algorithm

Figure 1: Overview of the forward computing dynamic slicing algorithms

Thesis Point 2: Connection between Version Control Operations and Maintainability

According to studies and experiences, code quality, especially code maintainability, has direct impact on development costs. On the other hand, the quality of the software source code declines if we do not invest to improve it. We were motivated to find out where and why this code erosion occurs: are there typical developer interactions causing similar changes in the maintainability? Knowing it might help preventing the code decay.

In this section we present the connection found between the number of operations in a version control commit and the maintainability change caused by that commit. First we summarize how we measured the maintainability of the source code. After that we present the actual new results: (2.A) how we discovered the existence of the connection, (2.B) what kind of impact of each version control operations we found, and finally (2.C) how we overcame the visualization problems.

Measuring Maintainability

We used the ColumbusQM for calculating the maintainability. This was published by Bakota et al. [1]. It is based on the ISO/IEC 9126 standard. The algorithm considers the following source code metrics: logical lines of code, number of ancestors, nesting level, coupling between objects, clone coverage, number of parameters, McCabe's cyclomatic complexity, number of incoming and outgoing invocations, and coding rule violations. Earlier studies state that the higher these values are (e.g. the longer a function is), the higher number of bugs it is likely to contain. It compares the metric values with other systems of a benchmark, and finally it aggregates the results.

2.A. Existence of the Connection between Version Control Operations and Maintainability

For every commit we determined the maintainability change, and based on this we partitioned them into three disjoint subsets: maintainability increase, no change, and decrease. On the other hand, we divided them based on version control operations into the following four categories: (D) commits containing file deletion; (A) commits not containing file deletion, but containing file addition; (U+) commits containing updates only, at least two; (U1) commits consisting of exactly one update operation. The combination of the two forms a matrix having 12 cells. Each cell contains the number of commits with matching conditions.

For every analyzed system we performed the contingency Chi-Squared test, which tells if the difference between the expected and the actual distribution is significant. We performed the analysis on four software systems: three of them were open source (Ant, Struts 2, and Tomcat), and one of them was an industrial one (Gremon).

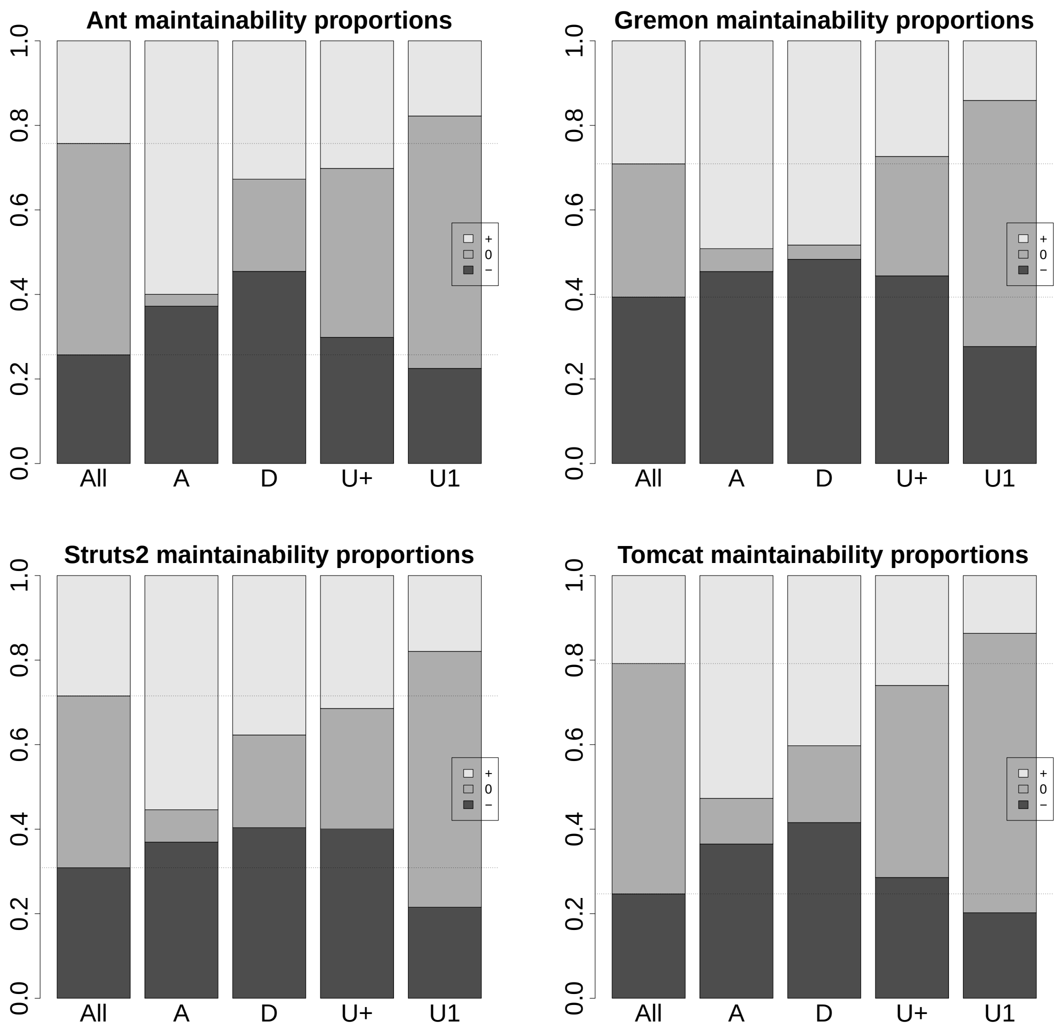

Figure 2: Maintainability proportions

Figure 2 shows a graphical overview of the data, where we illustrate the proportions of each commit category on bar plot diagrams. The bars with different shades indicate the proportions of the positive (light gray), neutral (gray) and negative maintainability change related commits (dark gray) for each category, and we display the overall proportion as well.

The test resulted a significant difference for almost all the cells. It deflected from the expected almost always in the same direction, and mostly with similar magnitude. Table 1 presents the overall resulting p-values.

System	**p-value**	**Significance**
Gremon	$1.19 \cdot 10^{-52}$	very strong
Ant	$1.60 \cdot 10^{-151}$	very strong
Struts 2	$4.47 \cdot 10^{-64}$	very strong
Tomcat	$4.84 \cdot 10^{-33}$	very strong

Table 1: Overall p-values of Chi-Squared tests

Based on these results we state that there is a connection between version control operation and the maintainability [14].

2.B. Impact of Version Control Operations on Value and Variance of Maintainability

We considered file additions, file updates and file deletions one by one, and checked their impact on the size [10] and the variance of maintainability change [5]. In the algorithm we divided the commits into subsets based on several aspects, and we compared the related maintainability change values. We considered the following divisions for all the three operations. First, we defined the main data set in three ways: (1) all commits, (2) commits containing the examined operation, (3) commits consisting exclusively of the examined operation. In each case we performed the following divisions: (a) division based on median of the number of that operation, (b) division based on the median of the proportion of that operation, (c) the main data set and its complementary. Therefore theoretically we defined 9 divisions, practically, by eliminating the trivial ones, we had 7 divisions, as follows (illustrated with operation Add):

DIV1: *Take all commits, divide them into two based on the absolute median of the examined operation.* It checks if commits containing high number of operation Add have better effect on maintainability than those containing low number of operation Add.

DIV2: *Take all commits, divide them into two based on the relative median of the examined operation.* It checks if the commits in which the proportion of operation Add is high have better effect on maintainability compared to those where the proportion of operation Add is low. To illustrate the difference between DIV1 and DIV2 consider a commit containing 100 operations, 10 of them are Addition (the absolute number is high but the proportion is low) and a commit containing 3 operations, 2 of them are Additions (the absolute number is low, but the proportion is high).

DIV3: *The first subset consists of those commits which contain at least one of the examined operations, and the second one consists of the commits without the examined operation.* It checks if commits containing file addition have better effect on the maintainability than those containing no file additions at all.

DIV4: *Considering only those commits where at least one examined operation exists, divide them into two based on the absolute median of the examined operation.* This is similar to DIV1 with the exception that those commits which do not contain any Add operation are not considered. This kind of division is especially useful for operation Add, as this operation is relatively rare compared to file modification. Therefore this provides a finer grained comparison.
DIV5: *Considering only those commits where at least one examined operation exists, divide them into two based on the relative median of the examined operation.* Similar to DIV2; see the previous explanation.
DIV6: *The first subset consists of those commits which contain the examined operation only, and the second one consists of the commits with at least one another type of operation.* This checks if commits containing file additions exclusively have better effect on the maintainability compared to those containing at least one non-addition operation. This division is especially useful in case of file updates.
DIV7: *Considering only those commits where all the operations are of the examined type, divide them into two based on the absolute median of the examined operation.* This division is used to find out if it is true that commits which consist of more file additions result in better maintainability compared to those consisting of less number of additions. It is especially useful in case of file updates, as most of the commits contain exclusively that operation.

So we performed the division of the commits on version control operation basis, and we considered the related maintainability change values. We defined the maintainability change values not simply as the differences of the subsequent revisions, but we considered their characteristic and the actual size of the code as well.

We compared these values. At the value comparison we used the Wilcoxon-test, which is not sensitive on the outliers. As a result, we gained an overview for each operations, in what circumstances what kind of effect they had on the maintainability.

We performed the analysis on the same input data as mentioned in Thesis Point 2.A. The test results showed that file additions improved, or at least less eroded the maintainability than file updates. The file updates mainly eroded them. We could not establish the effect of the file deletion; we received contradictory results.

In Figure 3 we illustrate the results of the comparisons visually. High absolute length of a bar means high significance within the analyzed software system. Positive value means positive connection between the actual version control operation and the maintainability.

We compared the variances belonging to the two sets of maintainability change values. The variance tests are extremely sensitive on the outliers, and the non-standard commits, like merging a branch into the master, might cause huge deflections. Therefore we omitted these commits at the comparison of the variances.

A spectacular method of comparing the variances is the ratio of variances, as illustrated in Figure 4. This figure contains the geometric means of the individual ratio values of each systems.

We concluded that the file addition and file deletion increased, while the file update decreased the variance. As the amplitude was much bigger than the absolute change, as a final conclusion we stated that it was recommended to pay special attention to file additions.

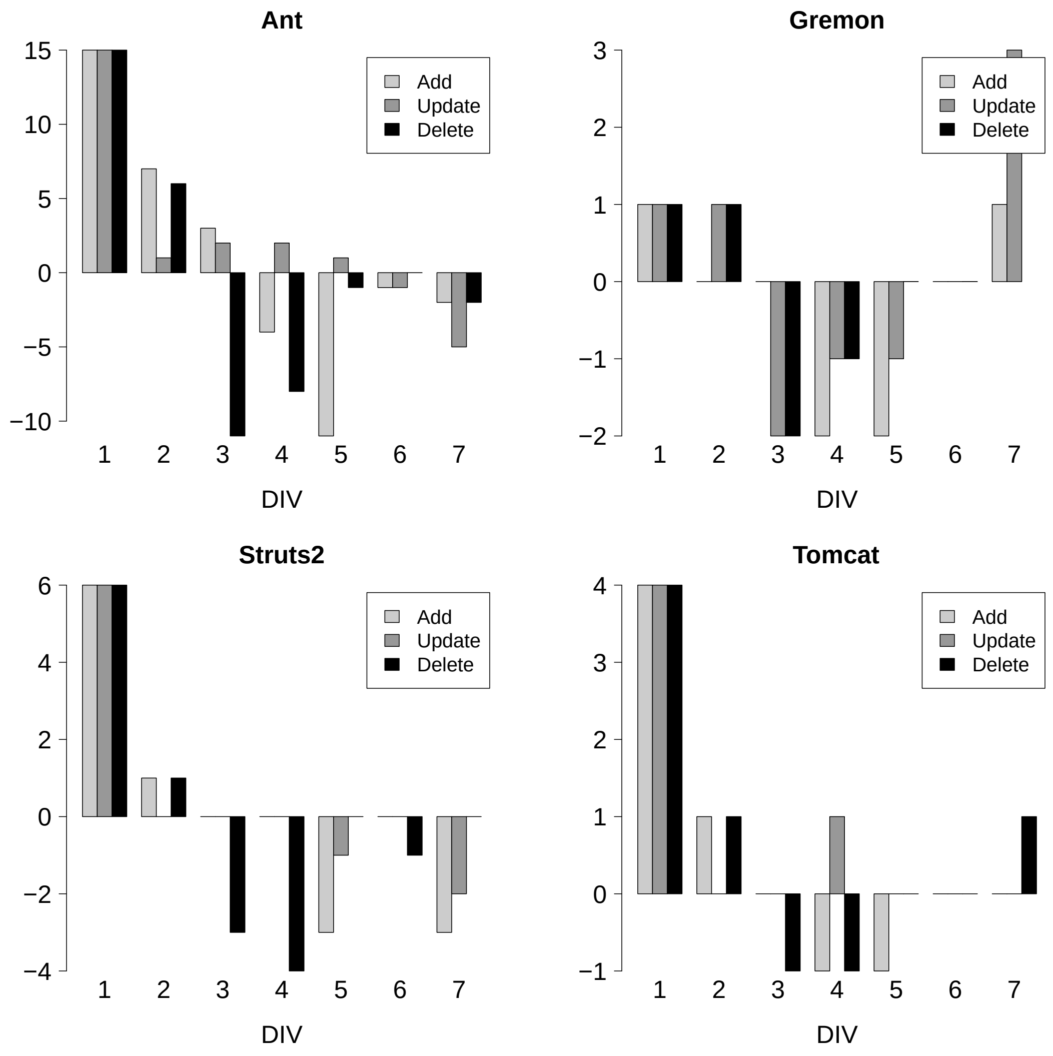

Figure 3: Wilcoxon test result bars

2.C. Cumulative Characteristic Diagram and Quantile Difference Diagram

We presented two visualization methods which were adequate for presenting some of the published results visually [6].

The input of the *Cumulative Characteristic Diagram* is a set of numbers. We sort them non-ascendant, and for every index we calculate the sum from the first one up to that point. The indices represent the x-coordinate, and the sums represent the y-coordinate. The characteristic is created by connecting these points with lines. On the Composite Cumulative Characteristic Diagrams we draw two or more Cumulative Characteristic Diagrams.

This diagram type turned to be suitable for illustrating the Chi-squared test, the Wilcoxon-test and the variance test. Figure 5 illustrates the input data used in Thesis Point 2.A. It considers the mentioned divisions, and creates CCD using related maintainability changes. The differences within the diagrams and similarities between the diagrams support the finding that connection between the number of version control operations and the related maintainability change exists. E.g. considering the smaller curves in the middle, the right end of the second one is always located above the right ends of the two curves on the right hand side, which indicates that the operation Add has better effect

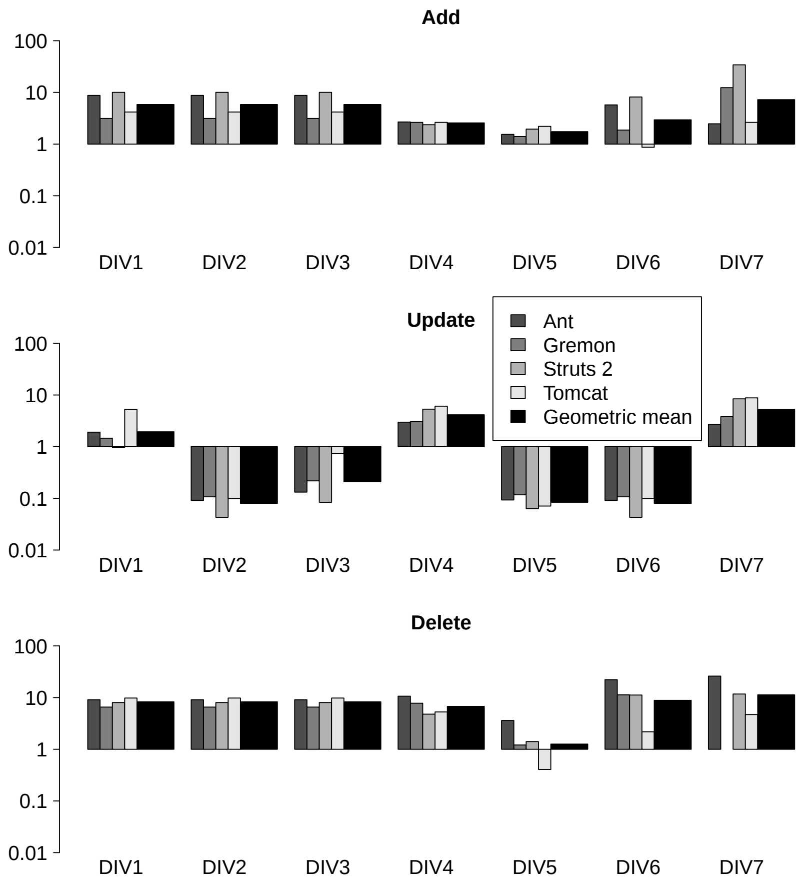

Figure 4: Illustration of variances

on the maintainability than operation Update. Another spectacular characteristic is that the width of the two left hand side curves are appreciably smaller than the width of the two right hand side curves, while their heights are mostly the same magnitude. This means the variances of the Delete and Add related maintainability changes are considerably higher than those values related to the two Update related subsets.

Using the *Quantile Difference Diagram* we can compare two sets of numbers. First we sort the elements of both sets in non-descending order. Then we take the values of every centile from both sets, pairwise. E.g. the median will be paired with the median, the 90% with the 90% from the other one etc. We calculate the difference of every pair. Using the centiles as the x-coordinate and the differences as y-coordinate, finally connecting the points we gain the Quantile Difference Diagram. If the original data contains outliers, it is recommended not to depict these values; the default implementation does not consider the lower and upper 5%.

This diagram type turned to be suitable for illustrating Wilcox test and variance test. Figure 6 illustrates one of the results of Thesis Point 2.B. It is the comparison of the maintainability change values related to commits containing vs. not containing file addition. The fact that major part of the line is located above the x-coordinate indicate commits containing file additions tend to cause better maintainability change, compared to those not containing file additions. The of the line slope indicates that the variance of the maintainability change between the two considered subsets are different [6].

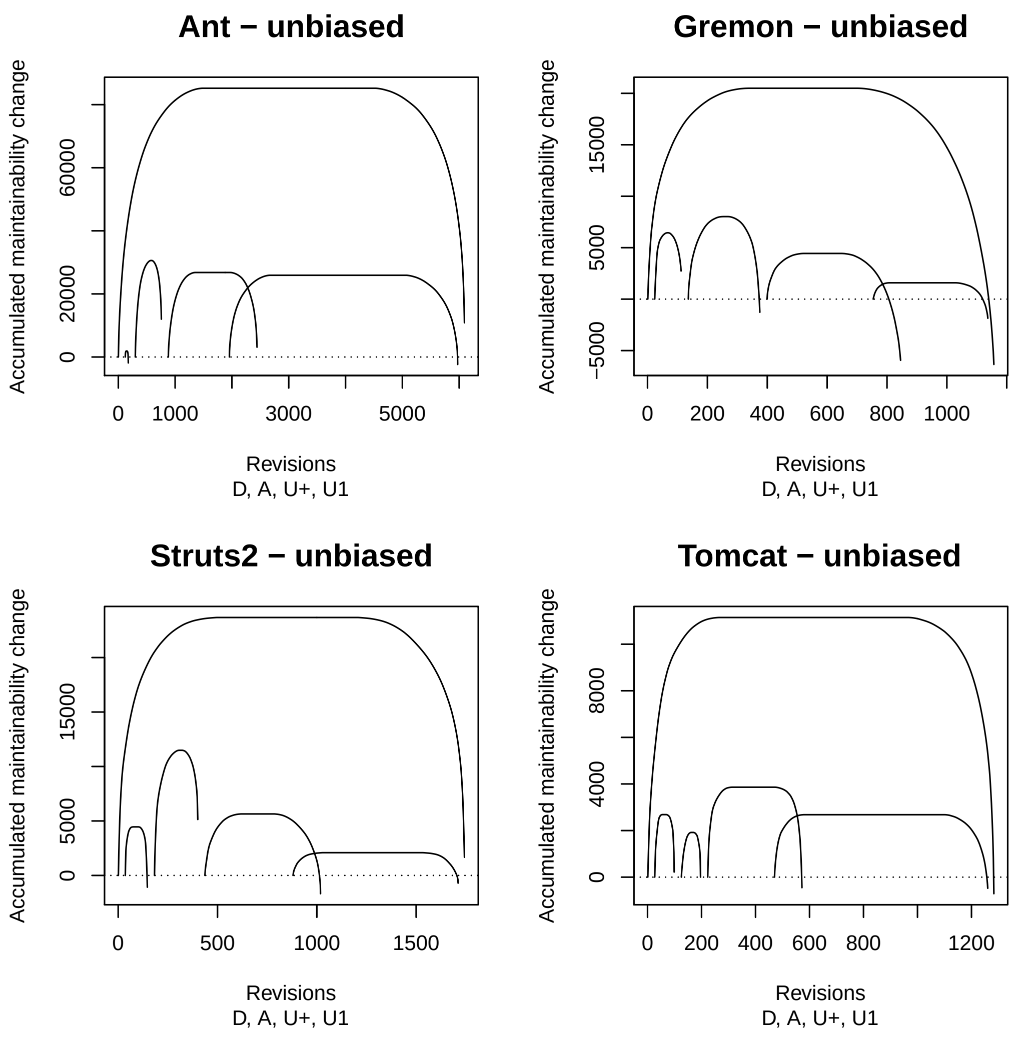

Figure 5: Composite cumulative characteristic diagrams about maintainability

The Author's Contributions to the Results

The author contributed to the new results presented in this section as follows:

- The methodology of finding the connection between version control operations and maintainability, along the implementation, the execution and the evaluation of the results. This methodology includes the following: the idea of categorization the commit on version control operation basis, using PCA; application of the Contingency Chi-Squared test on *maintainability change* and *version control operation based commit category* matrix. The evaluation of the results include the idea of interpretation of the results using the exponents, visualized by bar plot diagrams.
- The methodology of examining the impact of version control operation on the maintainability, along with the implementation, the execution and the evaluation of the results. The methodology includes the 7 divisions of commits on version control operation basis, and application of the Wilcoxon-test and the F-test on these subdivisions. The evaluation of the results include the idea of calculating with the standard residuals, and visualization of them with bar plot diagrams.
- The idea of the Cumulative Characteristic Diagram and Quantile Difference Diagram, implementation in R, maintenance of the `vudc` R package [7].

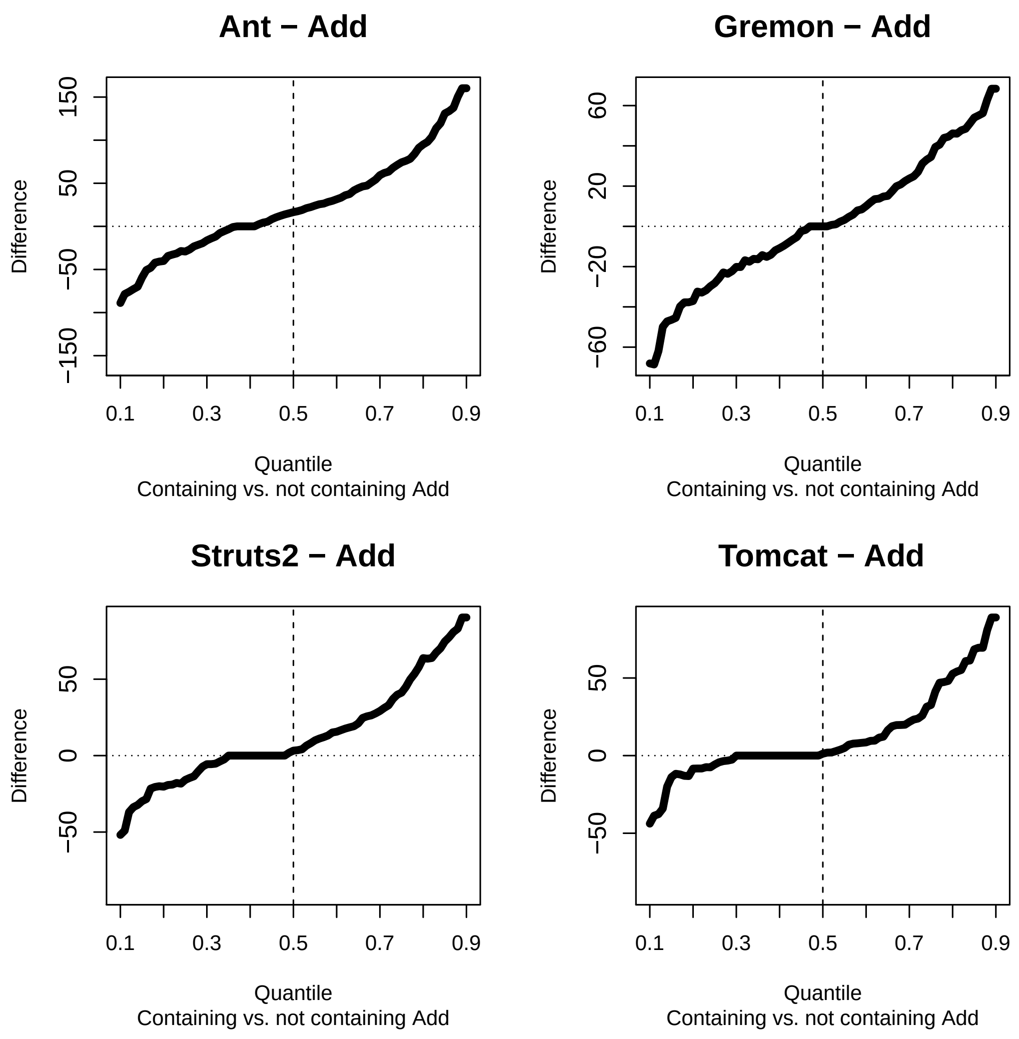

Figure 6: QDD about maintainability changes of commits with and without file additions

Thesis Point 3: Connection between Version Control History Metrics and Maintainability

We went further in analyzing the information located in version control systems. Unlike earlier, here we considered which piece of information was related to which file, therefore making a connection between different commits.

First we examined the effect of the intensity of past modification intensity of source code, and of the level of code ownership, on the later maintainability changes. After that we defined six version control history metrics, and considering all of them one by one we checked their connection with the maintainability, and, as a cross-check, with the number of post release bugs.

3.A. Impact of Code Modifications and Code Ownership on Maintainability

We checked the effect of past cumulative code churn [12] and the number of contributors [11] on the maintainability of the present commits.

We calculated for each file and revision from the very beginning, how many lines had been added and deleted all together. On a certain commit we averaged these values. We divided these values into two subsets, based on the maintainability change of the related commit, if it decreased or increased it (we omitted the commits related to neutral maintainability changes). Finally we compared the values using Wilcoxon-test.

We performed similar steps in case of code ownership analysis. There we checked how many different developer contributed to the file, and at certain commit we took their geometric mean. For the comparison here we also used Wilcoxon-test.

We performed the analysis on the input data used in Thesis Point 2.A. As the result – as presented in Table 2 – we gained that the past intensive modifications and the lack of clean code ownership foretold the decrease of the maintainability.

	Cumulative Code Churn		**Code Ownership**	
System	**p-value**	**Significance**	**p-value**	**Significance**
Ant	0.00235	very strong	0.03347	strong
Gremon	0.00436	very strong	0.05960	significant
Struts 2	0.00018	very strong	0.00001	very strong
Tomcat	0.03616	strong	0.21384	not significant

Table 2: Results of the cumulative code churn and code ownership tests

3.B. Correlation between Version Control History Metrics and Maintainability

We defined six version control history metrics, and checked their connection with maintainability [13]. These metrics were the following: cumulative code churn, number of modifications, ownership, ownership with tolerance, code age, and last modification time.

For a certain version of the analyzed system we sorted the source files based on every metrics. This resulted six orders of source files.

We determined the order of files on the Relative Maintainability Index [16] basis. The base idea in a nutshell is the following: the maintainability analysis is performed on the whole system, and on the system without the analyzed source code element. The Relative Maintainability Index is the difference between the original maintainability value and the maintainability value without that source code element.

As a cross-check, we determined the order of the source files based on the number of post-release bugs per source file as well.

Then we tested the similarity of these orders with help of the Spearman's rank correlation test. We performed the analysis on the following open source software projects: 5 version of Ant, 4 versions of jEdit, 3 versions of Log4J and 2 versions of Xerces; all together we analyzed 14 versions of 4 systems.

Table 3 contains the results of comparisons of the orders determined by each defined metrics and the Relative Maintainability Index. According to the results can state that the higher intensity of modifications, the higher number of code modifications and developers (without and with tolerance), the older code and the later last modification date resulted lower maintainability and higher number of post-release bugs.

System	Version	Churn	Modifs.	Ownership	Own.tolerance	Added	Modified
Ant	1.3	-0.861	-0.598	-0.392	-0.556	0.239	-0.563
	1.4	-0.867	-0.656	-0.475	-0.609	0.339	-0.373
	1.5	-0.747	-0.631	-0.550	-0.628	0.269	-0.592
	1.6	-0.852	-0.719	-0.636	-0.704	0.276	-0.464
	1.7	-0.702	-0.612	-0.560	-0.532	0.279	-0.268
jEdit	4.0	-0.712	-0.506	NA	-0.160	0.098	-0.442
	4.1	-0.681	-0.552	-0.515	-0.461	0.105	-0.466
	4.2	-0.713	-0.505	NA	-0.103	0.091	-0.478
	4.3	-0.302	-0.570	-0.488	-0.553	0.226	-0.044
Log4J	1.0	-0.823	-0.351	NA	-0.055	0.221	-0.283
	1.1	-0.873	-0.779	-0.556	-0.504	0.227	-0.535
	1.2	-0.854	-0.410	-0.481	-0.362	0.167	-0.102
Xerces	1.3	-0.660	-0.468	-0.217	-0.430	0.069	-0.100
	1.4	-0.481	-0.523	-0.322	-0.455	0.151	-0.355

Table 3: Spearman's correlation ρs of RMI comparison

The Author's Contributions to the Results

The author contributed to the new results presented in this section as follows:

- Elaborating the methodology of code churn and code ownership analysis, performing the statistic tests, and evaluating the results.
- Defining the six version control history metrics. Implementing the version control history metrics collector. Elaborating the methodology of correlation test with Relative Maintainability Index and post-release bugs, implementing it, executing the tests and evaluating the results.

Summary

In this thesis we are concerned with two major topics: the program slicing and the maintainability analysis.

The research area of *program slicing* is huge and mature. To summarize the results presented in this thesis in one phrase could be the following: it is related to the unstructured statements handling of a specialized version of a certain dynamic program slicing algorithm. Therefore the work presented in this thesis is like a cog in the machine.

The topic of the *connection between version control history and software maintainability* is, strictly speaking, a young research area. This research field is located between software maintainability and mining software repositories.

The area of software maintainability is the elder and bigger one, having about half of the most relevant articles appearing before the millennium. The studies typically consider source code metrics and bug prediction. The field of mining software repositories is a more recent and evolving one; the vast majority of the articles appeared after 2000. A few of them also deal with bug prediction.

We are not aware of any publication which deals with the connection between version control data and the values a software maintainability model calculates. Therefore the second part of the thesis can be considered as a new and still small field, but with big potentials.

First we opened the box and showed that connection between version control history data and software maintainability exists. At the beginning we considered version control operations only, but later we calculated also with other data as well, like file name, name of the developer or the date of the commit.

We think there are still big potentials left in this research field. First of all, the version control history data are so far not fully exploited: there are still several metrics left to be defined and analyzed. Aggregating the values – similarly to the quality models which aggregate source code metrics data – is still a fully open task.

To summarize, in the first part we made a small step forward in a big research area, while in the second part we did a pioneer work in a young research area.

In Table 4 we summarize the publications related to each thesis point.

	[8]	[9]	[2]	[14]	[10]	[4]	[5]	[6]	[12]	[11]	[13]	[16]
1.	●	●	●									
2.A				●								
2.B					●	●	●					
2.C								●				
3.A									●	●		
3.B											●	●

Table 4: Thesis points and supporting publications

Acknowledgments

I would like to thank my supervisor Dr. Rudolf Ferenc for his magnanimous help he provided for me remotely during my individual preparation. I also thank Dr. Péter Hegedűs on his fruitful virtual cooperation. The cooperation was excellent with my casual co-authors as well: Dr. Tibor Bakota, Gergely Ladányi and Ádám Zoltán Végh.

More than a decade earlier in the topic of program slicing we had a a great common work with Dr. Árpád Beszédes, Dr. Tamás Gergely and Zsolt Mihály Szabó, under the supervision of Dr. János Csirik and Dr. Tibor Gyimóthy. I gained very useful programming experiences with them.

My working place, the Lufthansa Systems Hungária Kft. supported some of my researches both financially and by free time. Bruce Derby revised the thesis grammatically. This could not have been done without the encouragement, patience and support of my wife Lilla. I thank everybody who helped my research activities!

Csaba Faragó, 2016

References

[1] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A probabilistic software quality model. In *Proceedings of the 27th International Conference on Software Maintenance (ICSM)*, pages 243–252. IEEE Computer Society, 2011.

[2] Árpád Beszédes, Csaba Faragó, Zsolt Mihály Szabó, János Csirik, and Tibor Gyimóthy. Union slices for program maintenance. In *Proceedings of the 18th International Conference on Software Maintenance (ICSM)*, pages 12–21. IEEE Computer Society, 2002.

[3] Árpád Beszédes, Tamás Gergely, Zsolt Mihály Szabó, Janos Csirik, and Tibor Gyimothy. Dynamic slicing method for maintenance of large C programs. In *Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 105–113. IEEE Computer Society, 2001.

[4] Csaba Faragó. Variance of source code quality change caused by version control operations. In *Proceedings of the 9th Conference of PhD Students in Computer Science (CSCS)*, pages 12–13, 2014.

[5] Csaba Faragó. Variance of source code quality change caused by version control operations. *Acta Cybernetica*, 22(1):35–56, 2015.

[6] Csaba Faragó. Visualization of univariate data for comparison. *Annales Mathematicae et Informaticae*, 45:39–53, 2015.

[7] Csaba Faragó. *vudc: Visualization of Univariate Data for Comparison*, 2016. R package version 1.1.

[8] Csaba Faragó and Tamás Gergely. Handling the unstructured statements in the forward dynamic slice algorithm. In *Proceedings of the 7th Symposium on Programming Languages and Software Tools (SPLST)*, pages 71–83, 2001.

[9] Csaba Faragó and Tamás Gergely. Handling pointers and unstructured statements in the forward computed dynamic slice algorithm. *Acta Cybernetica*, 15(4):489–508, 2002.

[10] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. The impact of version control operations on the quality change of the source code. In *Proceedings of the 14th International Conference on Computational Science and Its Applications (ICCSA)*, volume 8583 Lecture Notes in Computer Science (LNCS), pages 353–369. Springer International Publishing, 2014.

[11] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Code ownership: Impact on maintainability. In *Proceedings of the 15th International Conference on Computational Science and Its Applications (ICCSA)*, volume 9159 Lecture Notes in Computer Science (LNCS), pages 3–19. Springer International Publishing, 2015.

[12] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Cumulative code churn: Impact on maintainability. In *Proceedings of the 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 141–150. IEEE Computer Society, 2015.

[13] Csaba Faragó, Péter Hegedűs, Gergely Ladányi, and Rudolf Ferenc. Impact of version history metrics on maintainability. In *Proceedings of the 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*, pages 30–35. IEEE Computer Society, 2015.

[14] Csaba Faragó, Péter Hegedűs, Ádám Zoltán Végh, and Rudolf Ferenc. Connection between version control operations and quality change of the source code. *Acta Cybernetica*, 21(4):585–607, 2014.

[15] Tibor Gyimóthy, Árpád Beszédes, and István Forgács. An efficient relevant slicing method for debugging. In *Proceedings of the Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 303–321. Springer International Publishing, 1999.

[16] Péter Hegedűs, Tibor Bakota, Gergely Ladányi, Csaba Faragó, and Rudolf Ferenc. A drill-down approach for measuring maintainability at source code element level. *Electronic Communications of the EASST*, 60:1–21, 2013.