Software Complexities: Do System and Change Complexity Relate?

Aldosary, Buthainah
b_aldosa@encs.concordia.ca

Concordia University
Montreal, Canada

1 Abstract

System complexity is a term that encompasses the internal interations of a software and describes them. As the number of these entities increase, the number of interactions would exponentially increase leading to an unmanageable complexity. On the other hand, change complexity encompasses the artifacts that make up a single change such as diffs, patches and revisions. To see how well these two measures relate, we extracted data from the Apache HTTP Server Open Source Project and calculated the correlations between pairs of metrics. Upon completion of the analysis, we found that system and change complexity metrics correlate poorly. This is due to the fact that they exclusively measure different things.

2 Introduction

Traditional complexity measures are normalized on or require a whole file or system and were not designed to measure the complexity of code fragments. Therefor, the complexity of a change can be very different from the complexity of the files or system that contain a change (e.g a change to many simple files might be as complex as a change to one difficult file). Change complexity is related to, "how hard is it to understand and review a change" vs system complexity, which measures for example, "how hard is it to understand and potentially modify the system". Traditional complexity measures do not really measure the complexity of the system but simply the size of the files that make up the system. As a result, they do not add any additional information beyond how large a file is. In contrast, all of the change complexity measures are based upon the notion that changes that are farther from each other or involve multiple entities are more complex than those closer together involving fewer entities. The goal of this paper is to evaluate the correlation between system complexity measures and change complexity measures of an evolving system.

We have the following research questions:

1. Does the difference of traditional complexity measures before and after a change correlate with change complexity measures?

2. Does the weighted sum of change complexity measures up to a version correlated with the traditional complexity measures at that version?

3. Do the traditional measures calculated on only the files that had changed at a certain commit correlate with the change complexity measures at the last 1000 changes?

The rest of the paper is organized as follows: Section 3 describes the background and motivation of this work. Section 4 describes the method and data used. Section 5 and 6 present the results and discussion as well as threats to validity. Section 7 is the conclusion and finally in section 8, we describe potential future work.

3 Background and Motivation

Traditional complexity metrics require the system as a whole and were not designed to measure the complexity of code fragments. McCabe's metrics directly measures the number of linearly independent paths through a program's source code. Syntactic complexity metrics that are exclusively based on the structure of the program and the properties of the text (for example, redundancy of operators and operands, as Halstead's metrics do), do not add any more information than lines of code do [1]. Furthermore, multiple studies have shown that these measures correlate very strongly with lines of code and may not provide additional information about the complexity of the system. [4, 5, 6]. As previously stated, measuring the complexity of a change differs from measuring an entire system. For example, a change to numerous simple functions across multiple files may be as difficult as changing the code contained within a single complex function [2]. In this section, we provide a background on the various measures of system and change complexity.

3.1 System (Traditional) Complexity Measures

Metrics Related to Lines of Code

1. **Source Line of Code(SLOC):** To define SLOC, we use the definition given by Conte [Conte 1986]: A line of code is any line of program text that is not a comment of blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements.
2. **Lines of Code (LOC):** Lines of code refers to the total number of lines in each source code file including comments and blank lines
3. **Blank:** Blank is a count of the number of blank lines.
4. **Com.l:** Number of lines that are exclusively comments (no code).
5. **Com.n:** Number of comments in the file (a comment can be multiline).

Metrics Related to Complexity

McCabe's Cyclomatic Complexity

McCabe's Cyclomatic complexity is one of the earliest complexity measures developed by Thomas J. McCabe in 1976. It directly measures the number of linearly independent paths through a program's source code. Any program can be represented as a graph with the simplest element being a flat series of statements with no conditions, loops or branches [1]. For a graph G with n vertices, e edges and p exit points, the complexity v is defined as follows:

$$v(G) = e - n + 2p \tag{1}$$

It is worth mentioning that the minimum value for the cyclomatic complexity metric is 1, which corresponds to the flat series of statements with no bifurcations or loops. Every additional region in the flow graph would increase the Cyclomatic complexity by one unit.

1. **Totcy** Total McCabe's cyclomatic complexity (sum of all functions)
2. **Maxcy** Maximum McCabe's cyclomatic complexity. (between all functions).
3. **Mincy** Minimum McCabe's cyclomatic complexity.
4. **Avgcy** Average McCabe's cyclomatic complexity.
5. **Medcy** Median McCabe's cyclomatic complexity.

   Halstead's Complexity Measures

   Halstead's complexity metrics rely on the idea that programs should be viewed as expressions of languages both programming and written. It relies on the premise that there are mathematically sound relationships among the number of variables, the complexity of the code and the type of programming language statements used.
6. **Hlen** Halstead's length.
7. **Hvol** Halstead's volume.
8. **Hlevel** Halstead's level.

9. **H.Men.D** Halstead's number of mental discriminations.

3.2 Change Complexity Measures

Since the goal of each measure is the same, to assess the complexity of a change, we expect some of our measures to be highly correlated. In the interest of parsimony, we will select the simplest set of measures that adequately captures change complexity.

1. **Files and diffs (Mods)**: We count the number of files contained in a commit. The more files that change, the larger the affect proportion of the system. Commits with a large number of files will likely be difficult to understand.

2. **Change blocks (Hunks)**: We measure the number of contiguous change blocks, or hunks, and the distance between these blocks in a modified file and sum them across the entire review. Contiguous changes are likely easier to review than changes that are further apart and that are potentially in multiple functions within a file.

3. **Indentation**: Hindle *et al.* [3] created a complexity measure that can be used to measure the complexity of the entire system or of a change by examining the level of indentation on source lines. We calculate their measure for reviews. They found that the strongest predictors of complexity were the sum and standard deviation of the indentation of source lines.

4 Methodology and Data

A quantitative research methodology was used for this empirical study where the objective was to develop models pertaining to our research questions. The choice of this method is key as it provides a connection between the empirical observation and mathematical expressions of the quantitative relationships. The data used for this study was provided by Dr. Peter Rigby, namely from the Apache HTTP Server Project written in C. All measures were calculated at a particular version (commit) and the data was retrieved from June 18, 1999 - March 3, 2011; that is during a 12 year time period. However, no further information is given if this data includes header files or not. SQL scripts were written to extract data of system and change complexity measure that answer each of the research questions using PostgreSQL 9.3. The resulting data was then imported into RStudio for further analysis. Finally, Spearman's rank correlation coefficient matrix was deduced to answer each of our research questions.

In figure 1 we illustrate a system which undergoes numerous changes. The vertical lines represent each version. During each version's development, the change complexity measures are assessed. In between versions, traditional complexity measures are computed. These are denoted by CCM and TCM respectively. To answer our research questions, for example at version 3, we want to see if $\text{CCM}(\text{V3}) = \text{TCM}(\text{V4}) - \text{TCM}(\text{V3})$ (2), $\text{TCM}(\text{V4}) = \text{CCM}(\text{V1}) + \text{CCM}(\text{V2}) + \text{CCM}(\text{V3})$ (3), finally we want to see if $\text{TCM}(\text{V3}) = \text{CCM}(\text{V3})$ for the last 1000 commits (4).

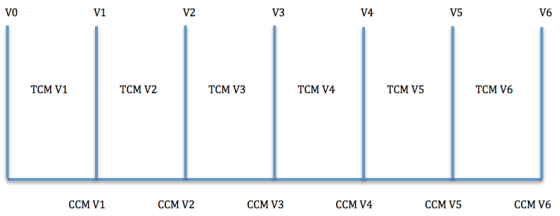

Figure 1: "Change Complexity vs. Traditional Complexity"

$$CCM_{v(i)} = TCM_{v(i+1)} - TCM_{v(i)} \tag{2}$$

$$TCM_{v(i+1)} = \sum_{i=0}^{i} CCM_{v(i)} \tag{3}$$

$$TCM_{v(i)} = CCM_{v(i)_{n=1000}} \tag{4}$$

5 Results and Discussion

It has been previously shown that Halstead's length, volume, level and mental discriminations all correlate highly with each other [1], so it is safe to select hlen only of these metrics. Maxcy and sloc have also shown to correlate highly with each

other so Maxcy has been chosen. Total cyclomatic complexity has been also added as it provides the complexity measure for the sum of all functions. Finally, loc has been included as it is a measure of all text lines in the code. For measures pertaining to change complexity, we have chosen the simplest ones that adequately capture change complexity.

***Q*1) Does the difference of traditional complexity measures before and after a change correlate with change complexity measures?**

We measure the difference between traditional complexity measures before and after a change and relate that with the change complexity measure at that particular change. By examining the result, we can see that measures that are related to change complexity correlate very poorly with those of system complexity (<0.3).

	mods	hunk_dist	indent_sum	loc	hlen	maxcy	totcy
mods	1.00	0.65	0.85	0.30	0.26	0.23	0.26
hunk_dist	0.65	1.00	0.63	0.18	0.16	0.11	0.14
indent_sum	0.85	0.63	1.00	0.28	0.24	0.22	0.24
loc	0.30	0.18	0.28	1.00	0.89	0.54	0.73
hlen	0.26	0.16	0.24	0.89	1.00	0.51	0.70
maxcy	0.23	0.11	0.22	0.54	0.51	1.00	0.64
totcy	0.26	0.14	0.24	0.73	0.70	0.64	1.00

Table 1: Correlation Matrix of Research Question 1.

***Q*2)Does the weighted sum of change complexity measures up to a version correlated with the traditional complexity measures at that version?**

We measured the last 1000 commits' weighted sum of change complexity measures up to a single version and compared it with the traditional complexity measures of that version. By looking at the results in table 2, we can see that change complexity measures correlated negatively with traditional complexity measures.

***Q*3)Do the traditional measures calculated on only the files that had changed at a certain commit correlate with the change complexity measures at the last 1000 changes?**

To answer the third research question, first we identified the files that have undergone changes in a version and extracted the traditional measures for them. Then, we measured the change complexity measures for the last 1000 changes and compared these two measures. Table 3 shows the correlation matrix which shows that there is a very poor correlation between the change complexity measures and traditional complexity measures (<0.3).

	mods	hunk_dist	indent_sum	loc	hlen	maxcy	totcy
mods	1.00	0.25	0.96	-0.23	-0.21	-0.38	-0.36
hunk_dist	0.25	1.00	0.31	-0.21	-0.22	-0.19	-0.20
indent_sum	0.96	0.31	1.00	-0.18	-0.16	-0.32	-0.30
loc	-0.23	-0.21	-0.18	1.00	1.00	0.95	0.96
hlen	-0.21	-0.22	-0.16	1.00	1.00	0.94	0.95
maxcy	-0.38	-0.19	-0.32	0.95	0.94	1.00	1.00
totcy	-0.36	-0.20	-0.30	0.96	0.95	1.00	1.00

Table 2: Correlation Matrix of Research Question 2.

	mods	hunk_dist	indent_sum	loc	hlen	maxcy	totcy
mods	1.00	0.66	0.86	0.23	0.23	0.21	0.23
hunk_dist	0.66	1.00	0.63	0.27	0.26	0.22	0.28
indent_sum	0.86	0.63	1.00	0.26	0.26	0.28	0.27
loc	0.23	0.27	0.26	1.00	0.99	0.82	0.96
hlen	0.23	0.26	0.26	0.99	1.00	0.85	0.96
maxcy	0.21	0.22	0.28	0.82	0.85	1.00	0.84
totcy	0.23	0.28	0.27	0.96	0.96	0.84	1.00

Table 3: Correlation Matrix of Research Question 3.

From the previous results of each of our research questions, we can see that syntactic complexity metrics don't provide more complexity information than lines of code do. We have also identified that traditional and change complexity measures correlate very poorly. This is due to the nature of how these measurements are extracted and what they concretely measure. For example, change complexity measures are extracted from diffs, revisions and patches as opposed to traditional complexity measures, which are extracted from the source code as a whole [3]. We can therefor explain the poor correlation with the underlying artifacts that these metrics are based on. Therefor, these complexity measures can not be substituted to be used as a proxy for the amount of effort that is needed to comprehend a piece of code.
A snapshot of the project from it's start date until it's end date is shown in the next figures. The traditional complexity measures is shown in Figure 2 in isolation, namely Locs, Totcy and Hlen. We can see that they all follow an increasing trend with Hlen and Loc decreasing at the same points in time. The change complexity measures (Mods, Hunk dist and Indent Sum) is shown in Figure 3 also in isola-

tion. They too follow a same trend with respect to each other. Finally, figure 4 shows the totCy and Hunk dist. In this case, we can see that they follow very different trends. This confirms that these two metrics indeed measure different things.

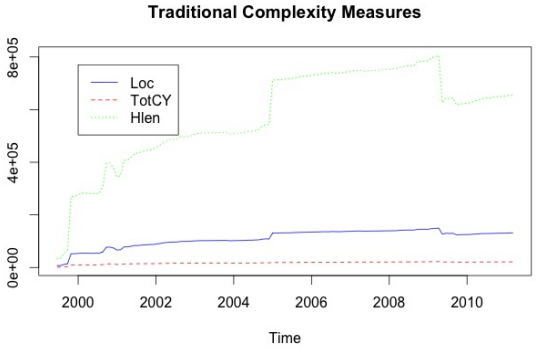

Figure 2: Traditional Complexity Measures Timeseries

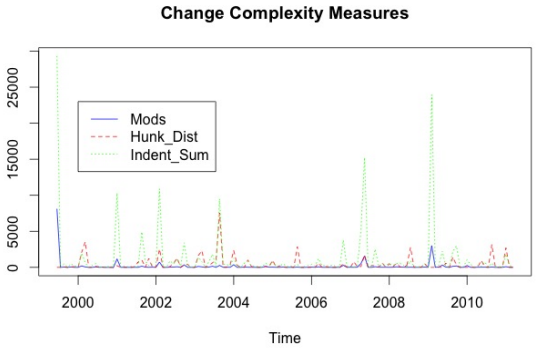

Figure 3: Change Complexity Measures Timeseries

6 Threats to Validity

In our study we calculated the complexity measures on each particular commits (aka. each version), however the correlation between traditional complexity measures in Israel's study [1] was done at a file level. Israel also made account of the confounding effect of file size. The result was that even though file size influences the values of the correlations, it does not substantially change the conclusion extracted from those correlations in any case. Therefor, in order to rule out this effect, the total size of the system must be taken into consideration. Another threat to validity is the source of inconsistencies in measuring traditional complexity metrics between header files and non-header files.

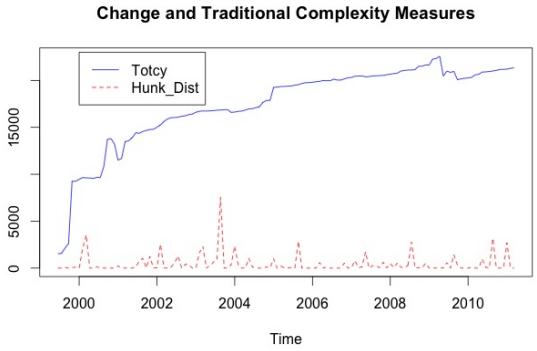

Figure 4: Traditional and Change Complexity Measures Timeseries

7 Conclusion

In this work we have chosen the Apache HTTP Server Open Source Project and extracted data related to traditional and change complexity measures within a 12-year period. For each of our research questions, we have shown that traditional complexity measures, in particular those that are exclusively based on the structure of the program or properties of text, do not provide additional information on the amount of effort that's required to fully understand a piece of code. Therefor, lines of code are sufficient to comprehend code fragments. We have also shown that, for each research question, while change complexity measures correlate very strongly with one another, the same result is not the case with traditional complexity measures (in some cases, there is a negative relationship). This is because they measure different things in different contexts and therefor results in a weak relationship between them.

8 Future Work

Future research directions include using traditional and change complexity metrics as an indication of the maintainability and evolvability of a system; such as productivity and number of bugs.

References

[1] I. Herraiz, A. Hassan *Beyond Lines of Code: Do We Need More Complexity Metrics?* Making Software. Greg Wilson and Andy Oram, O'Reilly Media, Inc., 2010.

[2] P. Rigby (2011), *Understanding Open Source Software Peer Review: Review Processes, Parameters and Statistical Models, and Underlying Behaviours and Mechanisms.*, Ph.D. Thesis. University of Victoria, Canada.

[3] A. Hindle, M. W. Godfrey, R. C. Holt. *Reading Beside the Lines: Indentation as a Proxy for Complexity Metrics.* In ICPC: Proceedings of the 16th IEEE International Conference on Program Comprehension, pages 133-142. IEEE Computer Society, 2008.

[4] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. *Predicting fault incidence using software change history.* IEEE Trans. Softw. Eng., 26(7):653-661, 2000.

[5] M. Leszak, D. E. Perry, and D. Stoll. *Classification and evaluation of defects in a project retrospective.* Journal of Systems and Software, 61(3):173-187, 2002.

[6] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. *Towards a theoretical model for software growth.* In MSR: Proceedings of the Fourth International Workshop on Mining Software Repositories, pages 21-28. IEEE Computer Society, 2007.