

How Open is Open Source? Metrics for measuring software mutation

Charles Hathaway

March 30, 2015

Abstract

Open source software is touted as being "openly accessible" to many people, thus allowing greater innovation and complex new methodologies. However, given the complexity of software and how difficult it is to design and implement, the question of whether or not secondary communities adapt and modify the software needs to be addressed. This paper will first summarize previous works regarding the structure of open source communities, discuss how useful or not useful previous attempts to quantify the "open-ness" of projects have been, and finally propose a metric for measuring how open a project is. We will conclude with a proposal for a technique to test the proposed metric.

1 Introduction

There have been many attempts over the years to measure software complexity; some of the most well known ones by McCabe [3], Oviedo [4], and Halstead [2]. In this article, we will not be proposing a complexity metric; rather, we will be proposing a way of measuring the change in complexity as projects modified and adopted by different groups. To do this, we will first create several use cases that maximize different metrics, then ask domain experts to help us determine if the represented code indicates a positive or negative change in complexity. For simplicity, this paper will utilize the visual programming environment created for the 3Helix program at Rensselaer Polytechnic Institute, CSnap.

2 Literature Review

2.1 A case study of the evolution of Jun: an object-oriented open-source 3D multimedia library

This paper discusses the development of an open source graphics library that focuses on ease of use over performance. Some notable things to consider; the project was written in Smalltalk (the same language as the original Scratch),

it still exists, and it was primarily developed by a small team (rather than a community)

?? uses graph theory to map developers to projects, and create a "graph" which they discuss in the form of network theory. This is very similiar to what I had hoped to do with Github, and the graph is fascinating. They coin the term "linchpin developer" to talk about developers who tie together projects. Very interesting paper, but it needs better formatting...

3 Discussion and comparison of previous metrics

In this section, we will summeraize and describe previous metrics considered. Weyuker wrote a paper in 1988[6] which considered many of these metrics, and we will therefore make references to her paper. For convience, I have copied the 9 properties of complexity measures she proposed here. Note, however, that they are explained in greater detail in the paper.

- Property 1: $(\exists P)(\exists Q)(|P| \neq |Q|)$
- Property 2: Let c be a nonnegative number. Then there are only finitely many programs of complexity c .
- Property 3: There are distinct programs P and Q such that $|P| = |Q|$
- Property 4: $(\exists P)(\exists Q)(P \equiv Q \ \& \ |P| \neq |Q|)$
- Property 5: $(\forall P)(\forall Q)(|P| \leq |P; Q| \text{ and } |Q| \leq |P; Q|)$
- Property 6a: $(\exists P)(\exists Q)(\exists R)(|P| = |Q| \ \& \ |P; R| \neq |Q; R|)$
- Property 6b: $(\exists P)(\exists Q)(\exists R)(|P| = |Q| \ \& \ |R; P| \neq |R; Q|)$
- Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statements of P , and $|P| \neq |Q|$
- Property 8: If P is a renaming of Q , then $|P| = |Q|$
- Property 9: $(\exists P)(\exists Q)(|P| + |Q| < |P; Q|)$

3.1 Cyclomatic complexity

Thomas McCabe proposed a complexity measure in his 1976 paper [3]. In this section, I will briefly describe how it works, how to calculate it, and discuss some responses to it (specifically the responses made by Elaine Weyuker in her 1988 paper [6]).

3.1.1 How it works

The McCabe metric (Cyclomatic number) analyses the control flow of a program to determine how complex it is. There are 3 primary items that is used the calculation of the cyclomatic number:

- Number of nodes, denoted as n
- Number of edges, denoted as e
- Number of components, denoted as p

Given these three variables, we can calculate a complexity (v) of a program using the following equation:

$$v = e - n + 2p$$

The real challenge is to create the graph so you can find values for these variables. To do this you must break a program into parts such that there is a starting node, and ending node, and a sequence of nodes between that two have the following key attributes:

- A compound statement is only one node; ie, `int i=0; i = b*x; ...` counts as only one node until the next branch occurs
- A branch is a conditional, either in the form of an if statement or a loop (with a loop, connect the node to both the following statement(s) and the code that gets executed after it exits)

As a general rule of a thumb, we have a component anywhere a loop occurs. We can have multiple components inside a component (embedded loops).

Once we have created a graph, it is a trivial matter of counting the number of edges, nodes, and components and applying the equation.

3.1.2 Analysis

This metric was included in the analysis performed by Weyuker in her 1988 paper [6]. Ultimately, she concluded that McCabes metric failed to address properties 2, 6, 7, and 9.

3.2 Normalized Compression Distance

[1]

3.3 Effort measure

[6]

3.4 Data flow complexity

Enrique Oviedo proposed a metric based on data flow in his 1980 paper [4]. This metric is based on 2 key components; data flow (DF) and control flow (CF). He concludes with the equation $C = \alpha CF + \beta DF$ (where $\alpha = \beta = 1$).

3.4.1 How it works

Control flow is simply the cardinality of the program flow graph. The program flow graph, and how to construct it, is clearly defined in the paper. It more or less follows the same structure as the graphs we construct for the Cyclomatic number, with a more formal definition being given to how a "program" (function in modern terminology) is denoted. Formally,

$$CF = \| E \|$$

Where $\| \quad \|$ stands for set cardinality.

The more complicated part of this metric is the data flow measure. The key terminology for this measure is the distinction between a locally available variable and a locally exposed variable. A variable is locally available when the variable is defined in the block (a block is a "node" in our control flow). A variable is locally exposed when it is referenced without being defined in the block. Another important note is that a variable can be "overridden" in a block if it is locally exposed, and then made locally available.

$$DF_i = \sum_{j=1}^{\|V_i\|} DEF(v_j)$$

Where V_i is the set of exposed variables in this block and $DEF(v_j)$ counts the number of available definitions for v_j .

$$DF = \sum_{i=1}^{\|S\|} DF_i + DF_f$$

Where S is all nodes except the start and the end nodes.

3.4.2 Analysis

This metric was included in the analysis performed by Weyuker in her 1988 paper [6]. Ultimately, she concluded that McCabe's metric failed to address properties 2 and 5.

3.5 Complexity Measure Based on Program Slicing (CMBPS)

[5]

4 Measuring change

There are several issues that need to be addressed by our algorithm if we are to effectively measure change in software over a large period of time. These include:

- Accounting for "non-changes"; ie, making a change to a project, testing it, then changing it back
- Discerning between changes that required little to no effort, and changes that may only have been to a line or two, but required a great deal of effort

- Considering things that change the output as part of the metric; not all changes will change the end-result of a program, but will rather optimize the process; other changes will cause the behavior of the program to dramatically alter. How do we differentiate these things?

5 Experiment proposals

In order to test our complexity measure, we will first apply it to the domain of education and student progress. The first step of entering this domain is to interface with the primary stakeholders; teachers and educators that work in the schools, with the primary audience; students. Rather than trying to "ask" opinions and talk in an abstract way, we have developed a system to measure the change in student work on our CSnap "community site", which teachers can use to help us find a relationship between the various metrics discussed and complexity as viewed by teachers.

Our first experiment will be used to help us calibrate the mutation measure against user input. We will design a number of scripts that each maximize a specific metric, while producing the same output, and ask teachers to rank them by complexity.

6 Conclusion

7 References

References

- [1] Rudi Cilibrasi and Paul MB Vitányi. "Clustering by compression". In: *Information Theory, IEEE Transactions on* 51.4 (2005), pp. 1523–1545.
- [2] Maurice H Halstead. "Elements of software science". In: (1977).
- [3] Thomas J. McCabe. "A Complexity Measure". In: *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING* SE-2.4 (1976), pp. 308–320.
- [4] Enrique I Oviedo. "Control flow, data flow and program complexity". In: *Software engineering metrics I*. McGraw-Hill, Inc. 1993, pp. 52–65.
- [5] Hongwei Tao and Yixiang Chen. "Complexity measure based on program slicing and its validation". In: *Wuhan University Journal of Natural Sciences* 19.6 (2014), pp. 512–518.
- [6] Elaine J. Weyuker. "Evaluating Software Complexity Measures". In: *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 14.9 (1988), pp. 1357–1365.