Cognitive Complexity

An Overview and Evaluation

G. Ann Campbell
SonarSource SA
Switzerland
ann.campbell@sonarsource.com

Abstract

As a measure of understandability, Cyclomatic Complexity is widely regarded as unsatisfactory, but until December 2016 it was the only one available. This paper describes Cognitive Complexity, a new metric designed specifically to measure understandability, and a brief survey of Cognitive Complexity issues in a subset of open source projects under static analysis on `SonarCloud`. From this analysis, an assessment is made of whether Cognitive Complexity is accepted or rejected by the developers of each project.

ACM Reference Format:
G. Ann Campbell. 2018. Cognitive Complexity: An Overview and Evaluation. In *TechDebt '18: TechDebt '18: International Conference on Technical Debt , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3194164.3194186

1 Introduction and Origins

In 1976, Cyclomatic Complexity was introduced to *identify software modules that will be difficult to test or maintain* [3]. While it accurately measures testability, most developers agree it fails at measuring understandability. After a series of discussions on addressing Cyclomatic Complexity's *understandability gap*, it was finally recognized at `SonarSource SA`, the company behind `SonarQube` and `SonarCloud` [1], that understandability cannot be measured by a strict mathematical approach. Thus Cognitive Complexity [2] was crafted to generate understandability ratings that tally with developer intuition to identify modules that will be difficult to maintain. This paper gives an overview of Cognitive Complexity, and examines developer reaction to it via a survey of Cognitive Complexity issues raised in open source projects.

2 How It Works

Cognitive Complexity is based on three rules:

1. Ignore structures that allow multiple statements to be readably shorthanded into one
2. Increment for each break in the linear flow of the code
3. Increment when flow-breaking structures are nested.

While the principles can be stated concisely, it is worth briefly enumerating the particulars.

2.1 Ignore readable shorthand structures

Readable shorthand structures are ignored to avoid penalizing developers for making code easier to read. Thus there is no increment for the method structure itself. Nor is there an increment for the null-coalescing operators found in many languages, such as `C#`'s `?.` and `??`.

2.2 Increment for breaks in the linear flow

With the theory that code could ideally be read like a novel; left to right, top to bottom in an unbroken flow; Cognitive Complexity increments when that flow is broken. Specifically, it increments once for each of the following:

- `switch`, `if`, `else if`, `else`, ternary operator
- `for`, `foreach`, `while`, `do while`
- `catch`
- `goto LABEL`, `break LABEL`, `continue LABEL`
- sequences of like binary logical operators
- each method in a recursion cycle.

2.3 Increment for nested control flow

Because Cognitive Complexity is an effort to measure the understandability of control flow, it must account for the fact that the difficulty increases as the control flow's context increases. Specifically, Cognitive Complexity is incremented commensurate with the level of nesting when control flow structures are nested.

2.3.1 Nesting level

The following structures increment the nesting level:

- `switch`, `if`, `else if`, `else`, ternary operator
- `for`, `foreach`, `while`, `do while`
- `catch`
- nested methods and method-like structures

2.3.2 Nesting increments

An increment is assessed for each level of nesting of the following structures inside structures listed in section 2.3.1.

TechDebt '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5713-5/18/05.
https://doi.org/10.1145/3194164.3194186

- `switch`, `if`, ternary operator
- `for`, `foreach`, `while`, `do while`
- `catch`

2.4 Exceptions

A few language-specific exceptions to these rules are enumerated in the Cognitive Complexity white paper [2], but these rules are generally applicable to all languages.

3 Developer Reaction

For a metric formulated to bridge Cyclomatic Complexity's *understandability gap*, the most important measure of success must be developer response.

3.1 Methodology

`SonarCloud` was used as a source of developer reaction. It offers free static analysis for open source projects, and includes analyzers with method-level Cognitive Complexity rules. Of the ten languages for which a Cognitive Complexity rule is available on `SonarCloud`, four were chosen as broadly enough used in open source to yield useful results: `Java`, `C#`, `JavaScript`, and `C++`.

The set of projects with relevant issues was narrowed to projects where Cognitive Complexity issues are actively managed, meaning that projects without at least one such issue marked either *False Positive* or *Won't Fix* were excluded. This criterion was used to ensure that the developers had taken a critial look at Cognitive Complexity in their projects.

Of the 8, 180 open source projects analyzed on `SonarCloud` by the time the data was collected on December 7-8, 2017, the selection process narrowed the set to 22 projects, plus one duplicate, which was ignored. Table 1 shows the projects and their distributions of issues across four resolutions: *Unresolved*, *Fixed*, *False positive*, and *Won't Fix*. *Unresolved* is the default, and issues are marked *Fixed* when they are no longer detected by the analyzer. *Won't Fix* and *False Positive* are only reached manually, and users are prompted for comment immediately after such interventions, although commenting is optional.

3.2 Analysis

Each project was categorized as either accepting or rejecting Cognitive Complexity based on an examination of its issues. The projects ultimately classed as rejecting are shown in Table 1 with a gray background.

An initial categorization was made based purely on issue status. Were all issues in the project either *False Positive* or *Won't Fix* (rejecting), or did the project include some *Fixed* or *Unresolved* issues (accepting)? That assessment was then refined based on comments made on rejected issues, and on the apparent circumstances under which the project's *Fixed* issues were closed. This process moved project 8 from rejecting to accepting based on its ameliorating issue comments. It also moved projects 1, 2, and 12 from accepting to rejecting based both on their dismissive issue comments, and on the apparently incidental means (one removed file and three fixed issues raised for complexity of only one above the threshold) by which those projects' few *Fixed* issues were closed. The categorization of the rest of the projects remained unchanged after this second pass.

This analysis yields a final tally of five rejections and 17 acceptances, for an acceptance rate of 77%.

Table 1. Survey Results. UN stands for Unresolved; FP for False Positives; F for Fixed and WF for Won't Fix.

ID	Project Organization / Name	UN	FP	F	WF
1	griffon / griffon	-	-	2	40
2	Caffeine / caffeine	-	-	1	40
3	Open Source / metaXplor-global	1	15	14	-
4	Open Source / Cubes	49	-	-	4
5	lastrix / ASN.1 Serialization	-	-	4	3
6	Frederik Kammel / FOK Launcher	4	-	3	3
7	Open Source / OmniFaces develop	19	-	-	3
8	vego1mar / Metody optymalizacji	-	-	-	2
9	Ivan / ticketsystem-mvg-app	-	-	-	2
10	Laurentius / Laurentius	65	-	3	2
11	Open Source / Bullwinkle	-	-	3	2
12	Open Source / go-bees develop	-	-	1	2
13	Groovylabs / Lyre develop	-	-	2	1
14	Alberto / Codec	1	-	-	1
15	Benjamin Bougot / Popcorn	26	-	14	22
16	Matteo Loro / epffanfic	2	-	-	1
17	vego1mar / Programowanie aplikacji w ASP.NET	-	-	1	1
18	Open Source / Review Board	14	-	-	48
19	choanioksource / reactChallengePleoIO master	-	1	1	-
20	Open Source / Valentina	-	-	470	66
21	Sebastian Schlag / KaHyPar	12	-	158	22
22	inilabs-github / libcaer	-	-	-	1

4 Conclusion

Given the fact that no coding guideline has universal approval, an acceptance rate of 77% is deemed a success. However, it is clear that the project sample size, and even the numbers of issues within the projects are too small to be statistically significant. More studies into the validity and utility of Cognitive Complexity are needed, and this is seen by `SonarSource` as an interesting topic for collaboration.

References

[1] [n. d.]. SonarCloud. https://sonarcloud.io. ([n. d.]). Accessed: 2018-03-06.

[2] G. Ann Campbell. 2017. *Cognitive Complexity - A new way of measuring understandability.* Technical Report. SonarSource SA, Switzerland. https://www.sonarsource.com/docs/CognitiveComplexity.pdf

[3] T. J. McCabe. 1976. A Complexity Measure. *IEEE Trans. Softw. Eng.* 2, 4 (Dec 1976), 308–320. https://doi.org/10.1109/TSE.1976.233837