Eyeballing the Cyclomatic Complexity Metric

Jurgen J. Vinju, Michael W. Godfrey

Centrum Wiskunde & Informatica, Amsterdam
David R. Cheriton School of Computer Science, Waterloo
Contribution

- “Control flow patterns” to analyze control flow
- Used to eyeball the CC metric
- Extremely simple
- Good for other research

Disclaimer: we introduce a metric to debunk another, not to introduce another metric
A “Control Flow Pattern” is an abstract syntax tree where all nodes that are not control flow statements are replaced by nothing. 

```
while (x >= 0) {
    if (x % 2 == 0)
        print("even");
    x--; 
}
return 1;

while (⊥) {
    if (⊥)
        ⊥
    x--; 
}
return ⊥;
```
A “Compressed Control Flow Pattern” is a control flow pattern where all empty lists have been removed and consecutive repetition has been contracted (based on structural equality), in a fixed point computation.

```c
switch (⊥) {
    case ⊥ : return ⊥;
    case ⊥ : return ⊥;
    case ⊥ : return ⊥;
    case ⊥ : return ⊥;
    case ⊥ : return ⊥;
    case ⊥ : return ⊥;
}
```
Control flow patterns matter

<table>
<thead>
<tr>
<th>Project</th>
<th>#Meth</th>
<th>#Pat</th>
<th>$#\text{Pat}^{\text{comp}}$</th>
<th>#Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>compendium</td>
<td>7,736</td>
<td>1,271 (16%)</td>
<td>1,234 (15%)</td>
<td>455 (36%)</td>
</tr>
<tr>
<td>Tomcat70</td>
<td>16,018</td>
<td>2,211 (13%)</td>
<td>2,158 (13%)</td>
<td>931 (43%)</td>
</tr>
<tr>
<td>dsbudget</td>
<td>306</td>
<td>64 (20%)</td>
<td>64 (20%)</td>
<td>18 (28%)</td>
</tr>
<tr>
<td>xml-external</td>
<td>3,346</td>
<td>91 (2%)</td>
<td>89 (2%)</td>
<td>30 (33%)</td>
</tr>
<tr>
<td>apache-ant</td>
<td>10,278</td>
<td>1,391 (13%)</td>
<td>1,349 (13%)</td>
<td>555 (41%)</td>
</tr>
<tr>
<td>bcel</td>
<td>3,076</td>
<td>286 (9%)</td>
<td>268 (8%)</td>
<td>120 (44%)</td>
</tr>
<tr>
<td>hsqldb</td>
<td>5,326</td>
<td>1,013 (19%)</td>
<td>969 (18%)</td>
<td>438 (45%)</td>
</tr>
<tr>
<td>smallsql</td>
<td>2,556</td>
<td>353 (13%)</td>
<td>332 (12%)</td>
<td>158 (47%)</td>
</tr>
<tr>
<td>Merged</td>
<td>48,642</td>
<td>5,633 (11%)</td>
<td>5,434 (11%)</td>
<td>2,455 (43%)</td>
</tr>
</tbody>
</table>
Cyclomatic Complexity

- Is defined on control flow graphs
- Measures the number of linear independent paths
- Estimates \( \# \) tests to cover code
- \( CC = \# \) conditions + 1

[wikipedia]
Measuring Understandability

Cyclomatic Complexity

Correlation with Badness
Measuring Understandability

Cyclomatic Complexity

Correlation with Badness
Measuring Understandability

Cyclomatic Complexity

Correlation with Badness
Measuring Understandability

Cyclomatic Complexity

Correlation with Badness
The Beef: Causality

- Does high CC cause bugs?
- Is high CC bad? Low CC good? Thresholds?
- What about CC-bug correlation in 5-10 years?
- Does CC work “in the small”, method by method?
Right?

- high CC
- many paths
- forget stuff
- make mistakes
- many bugs

correlation
Not the whole story

i = 0;
goto body;
loop:
  if (i == 10)
    goto done;
i++;
body:
  print(i);
goto loop;
done:

i = 0;
do
  print(i);
while (i++ != 10);
More than bargained for

```java
switch(state) {
    case A: return aap();
    case N: return noot();
    case M: return mies();
    case V: return vuur();
    case G: return gijs();
    case H: return hok();
    case D: return does();
    case S: return schaap();
    // and 50 more
}
```
Alternative story

bad programmer → can not find abstractions → accidental code

high CC → correlation → bugs
Alternative story

bad programmer → can not find abstractions → accidental code

high CC

bugs

correlation

DUH!
To say high CC causes bugs is not like saying high cholesterol causes cardiovascular disease.

It’s like saying that driving a BMW causes more visits to personal therapists.
Question

Does all that matter in real software?

That the CC metric both under and over appreciates the complexity of control flow?
Eyeballing CC

We assume that to appreciate the full complexity of the control flow of a method, you need to know all of it: all nodes and all edges.

Does CC predict the size of the full control flow of any given method or not?

Control flow patterns to correct for skew caused by frequently used programming idioms
Can CC predict size of control flow?

The CCFP for the above example would be:
```
switch (?) {
    R (case ? : return ? ;)
}
```

We can trivially extend the CFC metric to CCFPs now:

The compressed control flow complexity (CCFC) of a method is the number of nodes in the CCFP of that method.

Note that for any method $m$, $\text{CCFC}(m) \leq \text{CFC}(m)$, but there is no such inequality for CC. The reason is that CCFC could be more than CC, as well as less.

Our hypothesis is now that there should be many methods that are highly compressible. If so, then we deduce that the CC metric overestimates control flow understandability often. Using the above definition of a CCFP we have reduced all the patterns of the systems in Table 2. This now allowed us to plot the relation between the sizes of compressed patterns and normal patterns, and their respective distribution patterns. From this we can see how often there is repetition and how much repetition there is in the control flow of real Java methods.

2.3.2 Results

In Table 2 we see that compression occurs in more than 40% of all the patterns. At the same time, the statistics show that compression does not collapse many patterns together. In Figure 5 we can read the compression per CFP. We see that compression happens often (on all sizes of patterns), and that compression rates can be high for all sizes. Smaller patterns, if they compress, more often compress a little than a lot. Larger patterns, of which there are a lot less, do compress extremely in many cases. We have plotted linear, quadratic and square root fits (using least squares) such that it is clearly visible that the data set clearly favors small compression rates (many dots are printed on top of each other). The square root fits best for this data set, confirming that compression is more effective on larger patterns.

We have looked up a number of the larger methods to see what code would compress. In the systems we found 6...
• CC does not predict the size of a complete control flow graph (at all).
• CC does not predict the size of a complete control flow graph (at all).

• The published linear correlation between CC and method size is contentious.
• CC does not predict the size of a complete control flow graph (at all).

• The published linear correlation between CC and method size is contentious.

• CC practically always underrepresents the complexity of understanding a method.
• CC does not predict the size of a complete control flow graph (at all).

• The published linear correlation between CC and method size is contentious.

• CC practically always underrepresents the complexity of understanding a method.

• Additional edges do matter in real Java
How many times do repeated patterns occur? How much repetition in patterns?

The following is an example of a nested compressed CFP, found in the smallsql system.

```
switch (?
) {
    R (case ?:
        switch (?)
        {
            R (case ?:
                return ?;
            }
        })
}
```

We simplified it here for presentation purposes, by leaving out the context around the switch and removing some irregular cases. The pattern was associated to a single method which interprets boolean expressions in SQL with a CC of 126. The CCFC of this method is 75, and the CC of the compressed pattern is 27 (as opposed to the shown pattern which has CCFC equal to 7 and CC equal to 3). The method dispatches on the types of the arguments of an expression with the outer switch, and then on the operator kind in the nested switch and computes the result of the expression via recursion. This is a simple design for an interpreter that is trivial to understand. In theory, the particular nested pattern may represent a quadratic amount of methods, depending on how many repetitions occur at each level. In practice, we found that it occurs only once in the systems that we investigated.

The method with the largest compressed size (CCFC) was found in compendium: 179. It has an original CC of 141, its CFC is 198 and the CC of the compressed pattern is still 119. This is the worst case compression rate we found for such larger methods. The code dispatches on key press events and directly implements the associated actions.

The control flow structure is governed mostly by nested if-then and if-then-else statements with an occasional nested switch. Since in many cases single outermost conditional span multiple pages of source text, it is difficult to see which parts of the code are mutually exclusive and which are executed in order.

What compression does to the distributions of sizes of patterns and their cyclomatic complexity is shown in Figure 6. Between compressed and uncompressed the distributions have the same general shape, but compressed patterns have a larger peak below the threshold of size 10. This is a significant observation since 10 is a common threshold with CC for labeling a method to be "bad". In other words, many patterns go from being "bad" to being "good" by eliminating repetitive structure.

2.3.3 Analysis

Many patterns are reduced by compressing repetition, and this leads to significant reductions in size of the patterns. The compression is most evident in the larger patterns, although there are comparatively few of them. We can conclude that compression may be used to filter large methods that are easy-to-understand patterns and perhaps even generated. However, there are so few of such larger patterns that we should not jump to the conclusion that CC is not a good way of finding hard-to-understand patterns.

For smaller patterns the compression may be less visible, yet it has significant effect on the interpretation of the metrics. Although smaller patterns are usually not compressed below 50%, the compression does affect the interpretation of the metrics via the commonly used threshold of 10. From this perspective we can learn that systems that are easy to understand because they have repetitive control flow structure...
How many times do repeated patterns occur? How much repetition in patterns?

Quite a lot and more aggressively for the larger patterns
How are the compressed patterns distributed?

Figure 6. Size and CC distributions with and without compression.
How are the compressed patterns distributed?

Many “complex” patterns move across the “magic” threshold...
public AstNode compress(AstNode body) = innermost visit(body) {
    case [*a, repeated([*n]), n, *b] => [*a, repeated([*n]), *b]
    case [*a, x, *c, x, c, *d] => [*a, repeated([x,*c]), *d]
    case block(repeated(n)) => repeated(n)
};
Cyclomatic Complexity is not good for indicating understandability
Cyclomatic Complexity is not good for indicating understandability

Control flow patterns are useful
Cyclomatic Complexity is not good for indicating understandability

Control flow patterns are useful

Watch out with linear correlation
Industry is starting to believe some of our results. Now, this is a time to be very very careful, because our models will be used to judge the past and predict the future.

Just because you can fit a plot with a power function does not mean you have a scale-free distribution. You can fit it with anything.

Just because least squares produces a nice linear fit does not mean that you have a linear relation.

Always show the plots.