

18-642: Avoiding Spaghetti Code

1/29/2018



© 2017, Philip Koopman

Carnegie
Mellon
University

Hacker DICTIONARY

<http://www.hacker-dictionary.com/terms/spaghetti-code>

spaghetti code

n. Code with a complex and tangled control structure, esp. one using many GOTOs, exceptions, or other 'unstructured' branching constructs. Pejorative. The synonym 'kangaroo code' has been reported, doubtless because such code has many jumps in it.

■ Anti-Patterns:

- Deeply nested conditionals
- “Switch” nesting
- High Cyclomatic Complexity (too many paths through the code)

■ Unstructured code leads to bugs

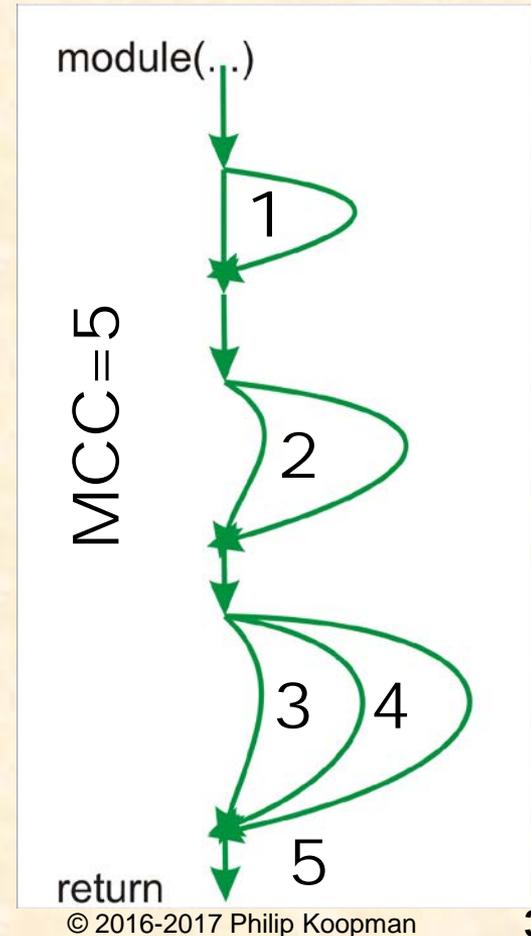
- Unstructured code is generally hard to understand, test, and review
 - But, even structured code can be problematic if it is too complex
- Want to limit complexity within each unit (e.g., subroutine, method)
 - Complex code is difficult to review – you will miss bugs during review
 - Complex code can be difficult or impossible to test

■ McCabe Cyclomatic Complexity (MCC)

- Measure each module (subroutine, method, ...)
- Draw a control flow graph
 - Graph has an arc for each path through the module
- MCC is # of “holes” in graph + 1
 - Worst case number of unit tests to cover all paths
 - Might need more tests – it’s a guideline

■ Strict Cyclomatic Complexity (SCC)

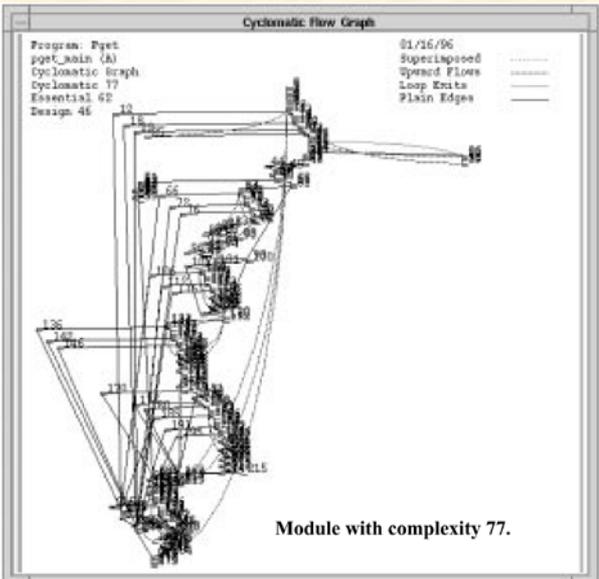
- For complex “if” tests, each condition counts
 - “if ((x == 0) || (y == 0)) ...”
counts as +2, because need to test $x \neq 0$ and $y \neq 0$
 - MCDC testing requires this type of coverage



High MCC Results in Tangled Code

■ Complexity beyond GOTO due to:

- Nested conditionals
- Overly complex lines of code
- Multiple return points
- Nested exceptions



■ Applying MCC

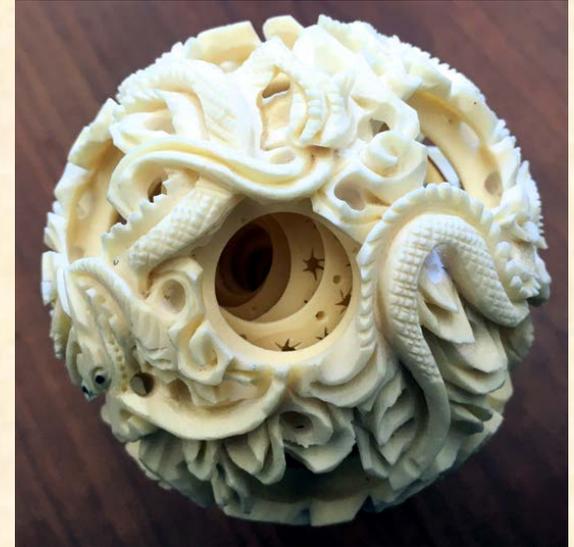
- Want maximum MCC to be 10 or 15
 - Above 30 is highly suspect
 - Above 50 is untestable in practice
 - » Too tangled to reasonably test each path
 - » Exception for flat switch statements
 - Above 75 predicts bug farms
 - » Each fix breaks something else

Spaghetti Factor (SF) Metric

$$\text{SF} = \text{SCC} + (\text{Globals} * 5) + (\text{SLOC} / 20)$$

- SCC = Strict Cyclomatic Complexity
- Globals = # of read/write global variables
- SLOC = # source lines of code (e.g., C statements)
- Scoring:
 - 5-10 - This is the sweet spot for most code
 - 15 - Don't go above this for most modules
 - 20 - Look closely; possibly refactor
 - 30 - Refactor the design
 - 50 - Untestable; throw the module away and redesign
 - 75 - Unmaintainable; throw the module and its design away; start over
 - 100 - Nightmare; throw it out and re-architect





■ Keep MCC below 10 to 15

- Even better, keep SCC below 10 to 15
 - Exception: easy to test flat switch statements are OK
- This enables thorough unit test

■ Additional signs of complexity issues

- “If” statements nested more than 2 or 3 deep
- Nested “if” and “switch” statements
- Excessive “break,” “continue,” multiple “return”

■ If your module is too complex, it’s time to break it up!

- Focus on worst offenders & break pieces of logic out into helper functions
- The point of this is to enable good peer review and good unit test

■ Complexity pitfalls:

- Creeping complexity over time ... at some point, refactor!

