Software Complexity Measurement

Software Engineering Principles

Week 25

1

Summary so far....

- No size estimation method is foolproof or particularly accurate
- Even once size is available, hard to extrapolate to effort, cost, estimated schedule, etc.
- Estimates can be self-fulfilling or self-defeating. Thus it is difficult to evaluate how well estimation is working, even retroactively
- Use an appropriate method for how much data you have – if no data, then gut instinct estimation is reasonable
- Try to avoid depending on your estimates being accurate

2

Therefore, let us try to keep it simple - KISS principle ☺

- **"The Future of digital systems is complexity, and complexity is the worst enemy of security."**

Bruce Schneier Crypto-Gram. Newsletter, March 2000

3

- **"The central enemy of reliability is complexity. Complex systems tend to not be entirely understood by anyone. If no one can understand more than a fraction of a complex system, then, no one can predict all the ways that system could be compromised by an attacker."**

CyberInsecurity Report
The Cost of Monopoly: How the Dominance of Microsoft Products Poses a Risk to Security
Computer & Communications Association
September 24, 2003

4

- **"Critical parts are typed up by hand and, despite a wealth of testing tools that claim to catch bugs, the complexity of software makes security flaws and errors nearly unavoidable and increasingly common."**
- **"The complexity will only increase as more business is automated and shifted onto the Internet and more software production is assigned to India, Russia and China."**

Forbes Technology , Saving Software From Itself Quentin Hardy, 03.14.05

5

Complexity Metrics

- The following metrics measure the complexity of executable code within procedures.
- High complexity may result in bad understandability and more errors.
- Complex procedures also need more time to develop and test.
- Complexity is often positively correlated to code size. A big program or function is likely to be complex as well. These are not equal, however.
- A procedure with relatively few lines of code might be far more complex than a long one.
- We recommend the combined use of LOC and complexity metrics to detect complex code.

Types of Complexity Metrics

Programming…

- ***Cyclomatic Complexity v(g)***
  - ***Comprehensibility***
  - ***Testing effort***
  - ***Reliability***
- Essential Complexity ev(g)
  - Structuredness
  - Maintainability
  - Re-engineering effort
- *Module Design Complexity iv(g)*
  - *Integration effort*

Data….

- Global Data Complexity gdv(g)
  - External Data Coupling
  - Structure as related to global data
- Specified Data Complexity sdv(g)
  - Structure as related to specific data

7

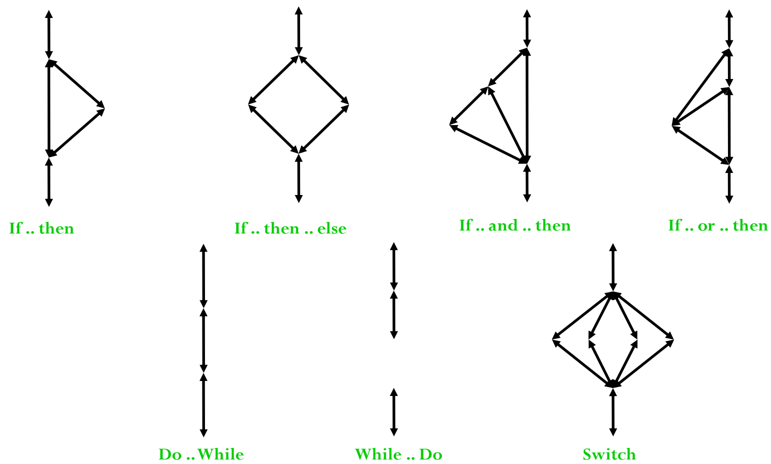
McCabe's Cyclomatic Complexity

- Definition: **Cyclomatic complexity** is a measure of the logical complexity of a module and the minimum effort necessary to qualify a module.
- **Cyclomatic** is the number of linearly independent paths and, consequently, the minimum number of paths that one should (theoretically) test.

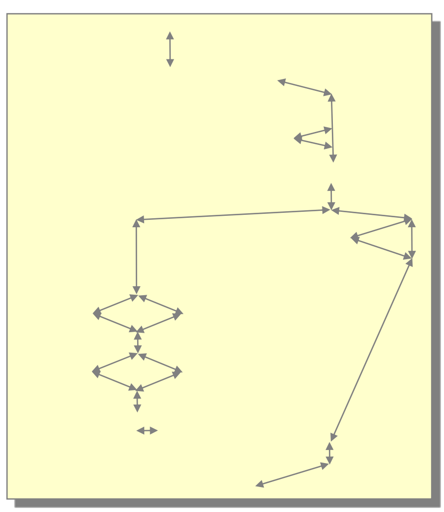
- **Quantifies the logical complexity**
- **Measures the minimum effort for testing**
- **Guides the testing process**
- **Useful for finding sneak paths within the logic**
- **Aids in verifying the integrity of control flow**
- **Used to test the interactions between code constructs**

8

Source Code Analysis Flow Graph Notation

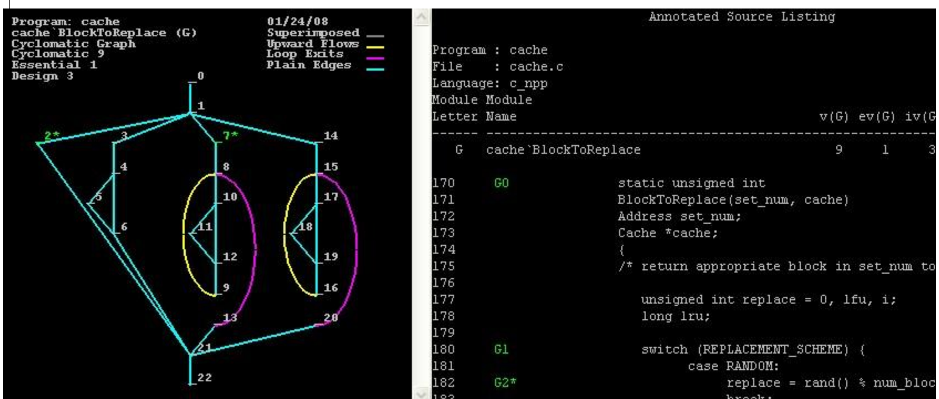


9



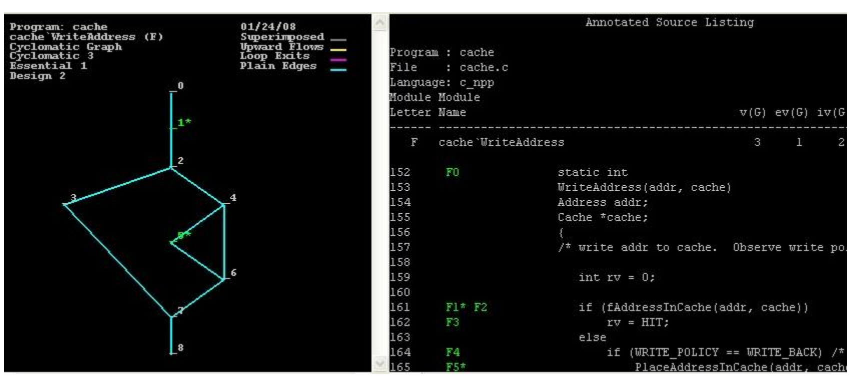
10

Complexity visualised with complexity measurement tools



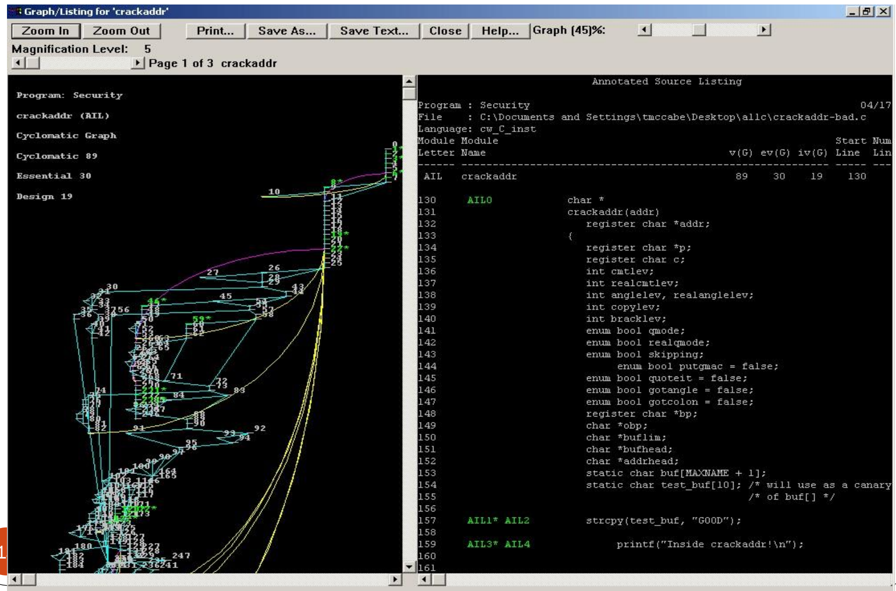
11

Something simple...



12

Something not so simple…



How to calculate Cyclomatic Complexity (Rule of thumb…)

- How to calculate cyclomatic complexity?
- *CC = Number of decisions + 1*
- What are decisions?
- Decisions are caused by conditional statements. In Visual Basic they are If..Then..Else, Select Case, For..Next, Do..Loop, While..Wend/End While, Catch and When.
- The cyclomatic complexity of a procedure with no decisions equals 1.
- A multiway decision, **the Select Case statement**, is counted as several decisions.
- This version of the metric does not count Boolean operators such as And and Or, even if they add internal complexity to the decision statements.

For ProjectAnalyser v8.0 – What is being added to CC?

- If..Then **+1** *An If statement is a single decision.*
- ElseIf..Then **+1** *ElseIf adds a new decision.*
- Else **0** *Else does not cause a new decision. The decision is at the If.*
- Select Case **+1 for each Case** *Each Case branch adds one decision in CC.*
- For [Each] .. Next **+1** *There is a decision at the start of the loop.*
- Do..Loop **+1** *There is a decision at Do While | Until or alternatively at Loop While | Until.*
- Unconditional Do..Loop **0** *There is no decision in an unconditional Do..Loop without While or Until.* *
- Catch..When **+2** *The When condition adds a second decision.*

Variations to Cyclomatic Complexity

- **CC2 Cyclomatic complexity with Booleans ("extended cyclomatic complexity")**
- *CC2 = CC + Boolean operators*
- CC2 extends cyclomatic complexity by including Boolean operators in the decision count.
- Whenever a Boolean operator (And, Or, Xor, Eqv, AndAlso, OrElse) is found within a conditional statement, CC2 increases by one.
- The reasoning behind CC2 is that a Boolean operator increases the internal complexity of the branch.
- You could as well split the conditional statement in several sub-conditions while maintaining the complexity level.

Variations to Cyclomatic Complexity

- **CC3 Cyclomatic complexity without Cases ("modified cyclomatic complexity")**
- *CC3 = CC where each Select block counts as one* CC3
- Equals the regular CC metric, but each Select Case block is counted as one branch, not as multiple branches.
- In this variation, a Select Case is treated as if it were a single big decision.
- This leads to considerably lower complexity values for procedures with large Select Case statements.
- In many cases, Select Case blocks are simple enough to consider as one decision, which justifies the use of CC3.

Relationship between CC and Risk

- A high cyclomatic complexity denotes a complex procedure that's hard to understand, test and maintain.
- **There's a relationship between cyclomatic complexity and the "risk" in a procedure.**

CC	**Type of procedure**	**Risk**
1-4	A simple procedure	*Low*
5-10	A well structured and stable procedure	*Low*
11-20	A more complex procedure	*Moderate*
21-50	A complex procedure	*alarming High*
>50	An error-prone, extremely troublesome, untestable procedure	*Very high*

Regardless of the exact limit, if cyclomatic complexity exceeds 20, you should consider it alarming.

Relationship between CC and Bad Fix Probability

CC	Bad fix probability
1-10	5%
20-30	20%
>50	40%
approaching 100	60%

Problems with McCabe's Complexity

- Although no one would argue that the number of control paths relates to code complexity, some argue that this number is only part of the complexity picture.
- **According to McCabe, a 3,000-line program with five IF/THEN statements is less complex than a 200-line program with six IF/THEN statements.**
- Another difficulty with the McCabe metric is that it seems too simple.
- **The McCabe metric does not reflect the well-known fact that programs are simple in some places, complex in others.**

Advantages…

- Despite these difficulties, the McCabe metric is an easy-to-compute, high-level measure of a program's complexity.
- In addition, the metric agrees with empirical data. McCabe and others found a high correlation between programs with high failure rates and high cyclomatic complexity.

Normalising CC

- **DECDENS Decision Density**
- Cyclomatic complexity is usually higher in longer procedures.
- How much decision is there actually, compared to lines of code?
- This is where you need decision density (also called cyclomatic density).
- *DECDENS = CC / LLOC*
- The denominator LLOC is the logical lines of code metric.

Total Cyclomatic Complexity (TCC)

- The TCC for a project or a class is calculated as follows.
- *TCC = Sum(CC) - Count(CC) + 1*
- In other words, CC is summed over all procedures.
- Count(CC) equals the number of procedures.
- It's deducted because the complexity of each procedure is 1 or more.
- This way, TCC equals the number of decision statements + 1 regardless of the number of procedures these decisions are distributed in.

DCOND: Depth of Conditional Nesting

- Depth of conditional nesting, or nested conditionals, is related to cyclomatic complexity.
- Whereas cyclomatic complexity deals with the absolute number of branches, nested conditionals counts how deeply nested these branches are.
- The recommended maximum for DCOND is 5.
- Although it might seem to give a lower DCOND, it's not recommended to join multiple conditions into a single, big condition involving lots of And, Or and Not logic.

Advanced Complexity Readings

- If you are really interested in structural complexity measures, there is a book that makes a thorough mathematical examination of 98 proposed measures for structural intra-modular complexity. This is for the very advanced reader.
- **Horst Zuse (1991) Software Complexity. Measures and Methods. Walter de Gruyter. Berlin - New York.**

Another Software Complexity Metric…

- *McCabe's measure (Cyclomatic complexity)*
- **Halstead's measures (computed statically)**
  - Program length: N = N1 + N2
  - Program vocabulary: n = n1 + n2
  - Volume: $V = N \times \log_2 n$
  - Difficulty : D = (n1/2) × (N2/n2)
  - Effort: E = D × V

Proposed Metric

- Definition

  Evaluating software complexity based on the following factors:

  - The number of variables defined and used in a software system
  - The number of arguments (parameters) involved in each function call in the source code

Motivation

A source code which utilizes more variables:

- **Intuition:** Is harder to understand, maintain, and migrate
- **Justification:** Considering most modern programming languages are strong typed, it requires more attention regarding:
  - Initialization
  - Type checking
  - Type conversion
  - Value tracking
  - Deconstruction

Motivation

- The same argument applies for function parameters
- Function arguments (parameters) can be considered variables
- Function is supposed to validate all parameters passed into it and handle all exceptions
- A function with a large number of parameters is hard to understand, debug, and maintain

Complexity Evaluation Rules

- **I.** In a statement that is not a variable declaration, each variable used will contribute one point to complexity index
- **II.** In a statement that is not a function declaration, each method/function invocation will contribute ***n*** points to complexity index (***n*** **=** the number of arguments)
- **III.** Class member attributes will be treated as variables except in their declaration statements

Validity

- Does the proposed metric really reflect the complexity degree of programs?
- Through a series of experiments and tests, the proposed measure was evaluated
- This was done by evaluating the complexity index and studying the difference between different revisions
- The revisions of the sample programs provide the same functionality, but the code is simplified

Case Study

- Testing the software tool and evaluating the proposed measure
- Merge sort and Quick sort are more complex than Bubble sort and Insertion sort
- Although the first two have better running times, but their source code is indeed more complicated

Case Study

Sort	Average	Best	Worst	Space
Bubble sort	O(n^2)	O(n^2)	O(n^2)	Can be in place
Insertion sort	O(n^2)	O(n^2)	O(n^2)	Can be in place
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Can be in place
Quick sort	O(n*log(n))	O(n*log(n))	O(n^2)	Can be in place

Sorting Algorithm	**Complexity Index**
Bubble Sort	20
Insertion Sort	19
Merge Sort	67
Merge Sort (Bad Implementation)	79
Quick Sort Standard Version	64
Quick Sort Improved Version	126

References

- S.L. Pfleeger, R. Jeffery, B. Curtis, B. Kitshenham. Status Report on Software Measurement. IEEE Software, March/April 1997.
- B. Clark. Eight Secrets of Software Measurement. IEEE Software, September/October 2002.
- J. Boegh, S. De Panfilis, B. Kitchenham, A. Pasquini, A Method for Software Quality Planning, Control, and Evaluation. IEEE Software, March/April 1999.
- J. Clapp. Getting Started on Software Metrics. IEEE Software, January 1993.

References (2)

- S.L. Pfleeger. Software Metrics: Progress after 25 Years?, IEEE Software, November/December 2008.
- R.J. Offen, R. Jeffery. Establishing Software Measurements Programs. IEEE Software, March/April 1997.