Automating Software Design Complexity Assurance

9th ICCC, Korea
Sept 24, 2008

Tim Huntley

Copyright © 2007 Juniper Networks, Inc.

Agenda

- Common Criteria complexity requirements
- Why care about complexity?
- Example complexity metrics
- Pros and cons
- Tools available
- Conclusions

Copyright © 2008 Juniper Networks, Inc.

Common Criteria Complexity Requirements

ADV_INT.3 Minimally complex internals

ADV_INT.3.1D The developer shall design and implement the entire TSF such that it has well-structured internals.

ADV_INT.3.2D The developer shall provide an internals description and justification.

ADV_INT.3.1C The justification shall describe the characteristics used to judge the meaning of “well-structured” and “complex”.

ADV_INT.3.2C The TSF internals description shall demonstrate that the entire TSF is well-structured and is not overly complex.

Copyright © 2008 Juniper Networks, Inc.

Common Criteria Complexity Requirements

Appendix A, A.3.2: Complexity of procedural software

"Complexity is the measure of the decision points and logical paths of execution that code takes….

Design complexity minimization is a key characteristic of a reference validation mechanism, the purpose of which is to arrive at a TSF that is easily understood so that it can be completely analyzed."

-- Part 3: Security assurance components, September 2007 Version 3.1 Revision 2, Appendix A, A.3.2: Complexity of procedural software.

Copyright © 2008 Juniper Networks, Inc.

Why care about complexity?

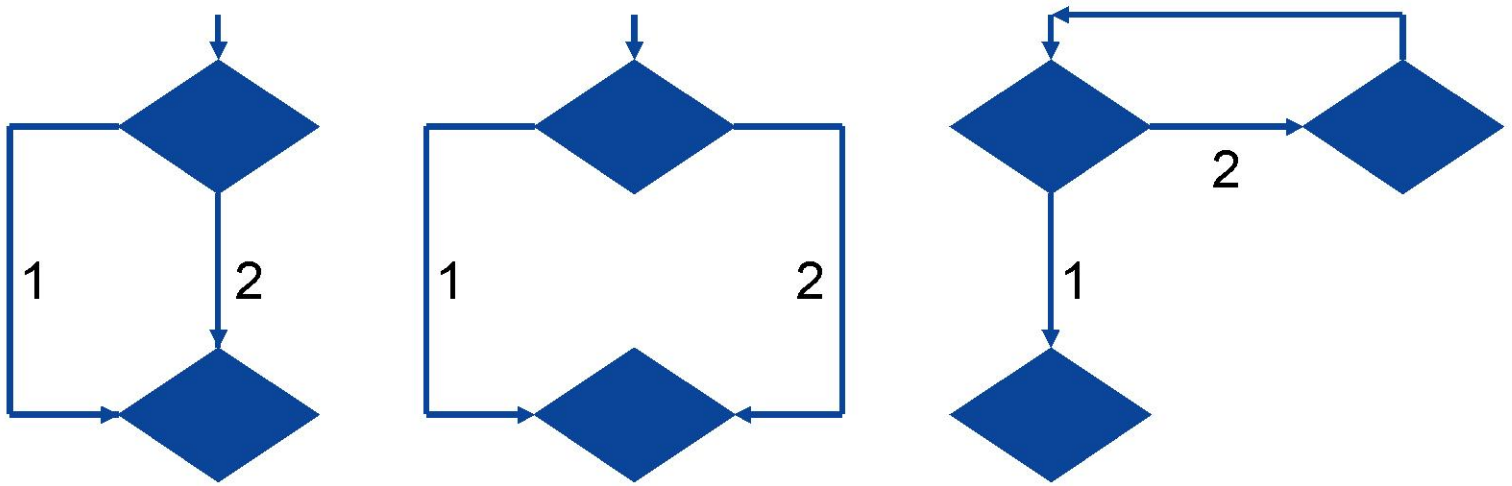
- ***Complexity correlates to higher bug count***
  - *High complexity generates "emergent behavior"*
- ***More complex software is harder to:***
  - *Understand*
  - *Maintain*
  - *Test*
- ***"Complexity is the worst enemy of security; as systems become more complex, they get less secure."***

  –Bruce Schneier, CSTO of British Telecom, author of *Secrets and Lies: Digital Security in a Networked World*

Copyright © 2008 Juniper Networks, Inc.

Cyclomatic Complexity

- *Introduced in 1976 by Thomas McCabe*
- *Expressed as single integer value*
- *Measures number of independent paths in a program*
- *Most popular complexity metric*



If-then If-then-else While/for loop

Copyright © 2008 Juniper Networks, Inc.

McCabe's Cyclomatic Number

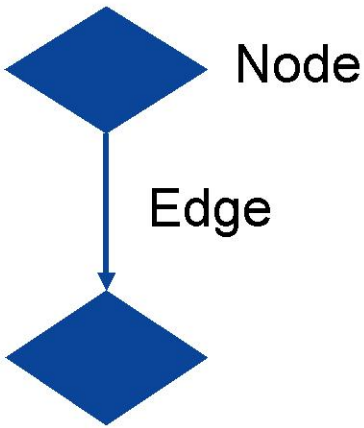
$$M = E - N + 2$$

where

M = cyclomatic complexity

E = the number of edges

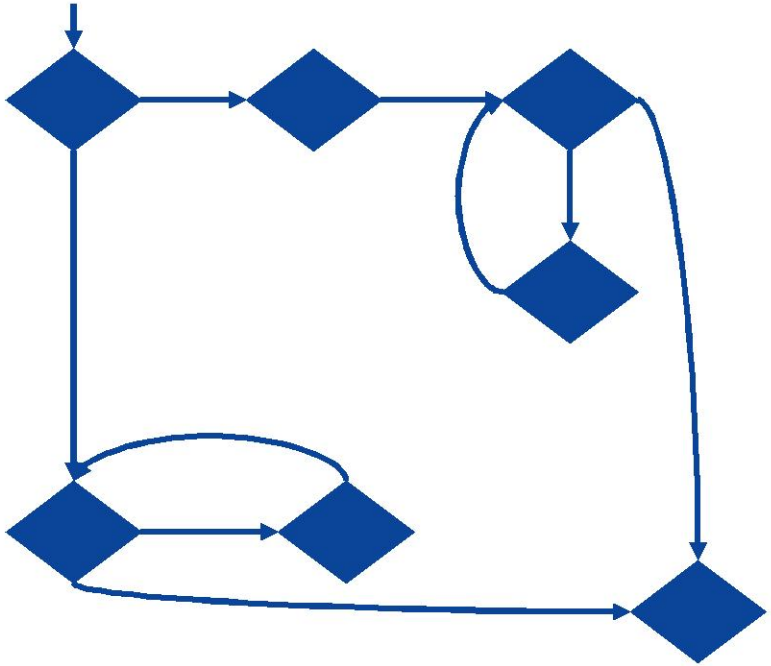
N = the number of nodes



Copyright © 2008 Juniper Networks, Inc.

Cyclomatic Complexity Example

```
void main()
{
if (location == Cheju) {
  findPool();
  while (wallet != empty)
    orderDrink();
}
else {
  while (eyes == open)
    readCCDocs();
}
return;
}
```



E = 9

N = 6

M = 9 – 7 + 2 = 4

Copyright © 2008 Juniper Networks, Inc.

CCM and structured testing

- M == number of paths through a module
- Each path requires one test case for complete code coverage
- Initial CCM can establish a base line for test cases
- A delta in the CCM can identify the need for new test coverage
- Records of CCM can be used to justify an assurance continuity argument

Copyright © 2008 Juniper Networks, Inc.

Risk Thresholds

Cyclomatic Complexity	Risk Threshold
1-10	Simple function, low risk
1-20	More complex function, moderate risk
21-50	Complex function, high risk
51+	Un-testable function, very high risk

Source: Carnegie Mellon Software Engineering Institute

Copyright © 2008 Juniper Networks, Inc.

CCM Tools

- **Commercial**
  - McCabe Software, http://www.mccabe.com/
- **Freeware**
  - CCCC: C and C++ Code Counter, http://cccc.sourceforge.net/
  - Source Monitor, multi-language analyzer, http://www.campwoodsw.com/sourcemonitor.html
  - CyVis, Java complexity analyzer, http://cyvis.sourceforge.net/
  - Open source analyzer for Perl, http://search.cpan.org/dist/Perl-Metrics-Simple/

Copyright © 2008 Juniper Networks, Inc.

Cyclomatic Complexity: Pros

- Simple: a single integer value is easy to interpret
- Objective: allows direct comparison between various designs
- Predictable: a developer can use it as early as the design phase
- Specific: indicates lower limits on test cases
- Usefulness not limited to certification

Copyright © 2008 Juniper Networks, Inc.

Cyclomatic Complexity: Cons

- **Simple: can easily mislead**
  - Example: a switch/case statement with many simple options generates a high complexity rating
- **Focused on code complexity; does not capture data complexity**
- **Correlation between error proneness and CCM not strong until CCM > 25**
- **Scope of analysis will affect results**

Copyright © 2008 Juniper Networks, Inc.

Halstead's complexity measures

- Introduced in 1977
- Focused on computational complexity
- Contrast with CCM which focuses on branch complexity
- Not as popular as CCM
- Lower correlation with fault proneness than CCM

Copyright © 2008 Juniper Networks, Inc.

Halstead's 5 measures

Measure	Symbol	Formula
Program length	N	N = N1 + N2
Vocabulary	n	n = n1 + n2
Volume	V	V = N * (Log2 n)
Difficulty	D	D = (n1/2) * (N2/n2)
Effort	E	E = D*V

n1 = number of distinct operators

n2 = number of distinct operands

N1 = total number of operators

N2 = total number of operands

Copyright © 2008 Juniper Networks, Inc.

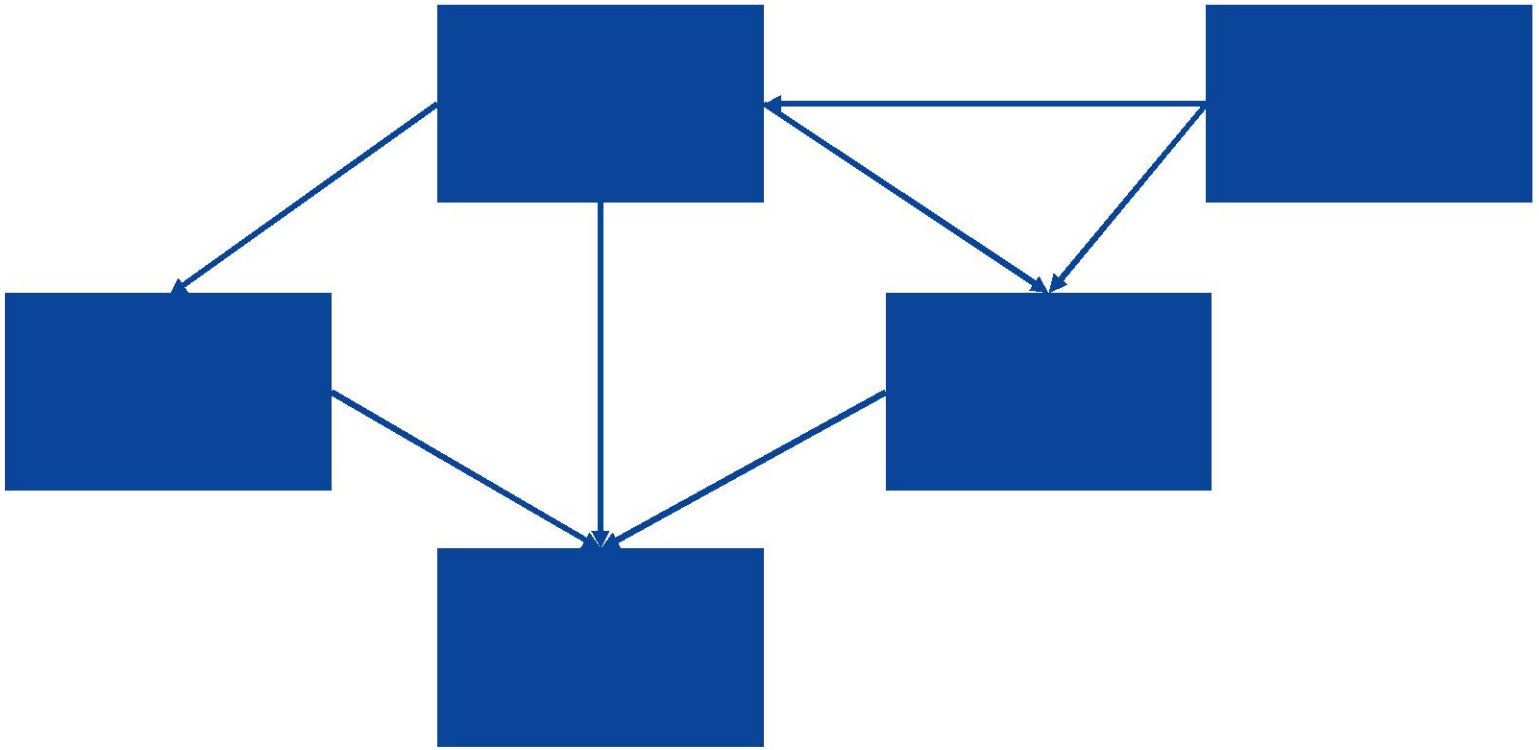
Halstead's complexity measures: Pro and Con

- **Pro:**
  - Captures complexity in calculational logic that is missed by CCM
  - Correlates with maintenance effort
- **Con:**
  - Lower correlation with fault proneness than CCM
  - Not easily used during the design phase
  - Ignores logic flow complexity

Copyright © 2008 Juniper Networks, Inc.

Henry and Kafura's metric

- Complexity = Length * (Fan-in * Fan-out)^2
- Measures coupling between modules



Copyright © 2008 Juniper Networks, Inc.

Henry and Kafura's metric: Pro and Con

- **Pro:**
  - Can be used during the design phase
  - Can be used with a large scope – multiple modules
- **Con**
  - Complexity = 0 if a module has no external coupling

Copyright © 2008 Juniper Networks, Inc.

Potential combinations

- **To capture internal module complexity + external coupling:**
  - Complexity = CCM * (Fan-in * Fan-out)^2
- **Carnegie Mellon SEI Maintainability Index:**

$$MI = 171 - 5.2 * ln(aveV) - 0.23 * aveV(g') - 16.2 * ln(aveLOC) + 50 * sin(sqrt(2.4 * perCM))$$

where:

aveV = average Halstead Volume (V) per module
aveV(g') = average CCM per module
aveLOC = the average lines of code (LOC) per module
perCM = average percent lines of comments per module

Copyright © 2008 Juniper Networks, Inc.

Conclusions

- Metrics are useful, objective tools for measuring complexity
- Intelligent interpretation is required
- Reliability increases toward the extreme end of the spectrum
- No single metric captures all facets of complexity

Copyright © 2008 Juniper Networks, Inc.

For more information

- Carnegie Mellon Software Engineer Institute, www.sei.smu.edu
- McCabe Software, www.mccabe.com
- NIST Special Pub. 500-235, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric

Copyright © 2008 Juniper Networks, Inc.

Speakers Contact Info

- **Tim Huntley**
  - Senior Software Engineer
  - thuntley@juniper.net

Copyright © 2008 Juniper Networks, Inc.

Copyright © 2007 Juniper Networks, Inc.