Software Engineering

COEN 174

Ron Danielson

Software Engineering Metrics

Chapter 8

Objectives

- Understand the characteristics of good design
- Learn legacy metrics to measure complexity of a design
- Learn OO metrics of design complexity
- Understand coupling and cohesion

Good Design

- "I know it when I see it"
- Many facets of a good design
  - Easy to understand
  - Easy to change
  - Easy to code from
  - Consistency
    - User interface
    - Error processing
    - Reports
    - System interfaces
    - Help

Good Design (cont.)

- Completeness (traceability tables can help)
  - All requirements are included
  - All parts of design are complete, to the same level of depth
- The "ities"
  - Portability
  - Maintainability
  - Understandability
  - Usability
  - Reusability

Good Design (cont.)

– The “easy tos”
  - Understand
  - Change
  - Reuse
  - Test
  - Integrate
  - Code

Why Metrics?

- Attempt to quantify qualities possessed by a software system
  - Which are often qualitative or subjective, and hard to measure
- Concreteness, even potentially illusive concreteness, makes comparisons possible
  - And thus helps improve the SW development process
- NOTE metrics must be easy to calculate, intuitive, and consistent

Legacy Metrics

- Many of these are code-based, not design-based
- Lines of code
  - Easy to calculate, consistent, intuitive
    - Longer programs have more functionality and are therefore more complex
  - Started being used with assembler programs, where relationship may be more valid
  - Comments and white space increase lines of code but may increase understandability
  - Short, tricky code segments can be most complex

Halstead’s Complexity Metric

- early 1970s
- Rationale: the more operators and operands a system has, the more complex it is
- Operators are things like +, -, if, while, …
- Operands are variables and constants
- Four measures
  - n1 = number of distinct operators
  - n2 = number of distinct operands
  - N1 = total number of operators
  - N2 = total number of operands

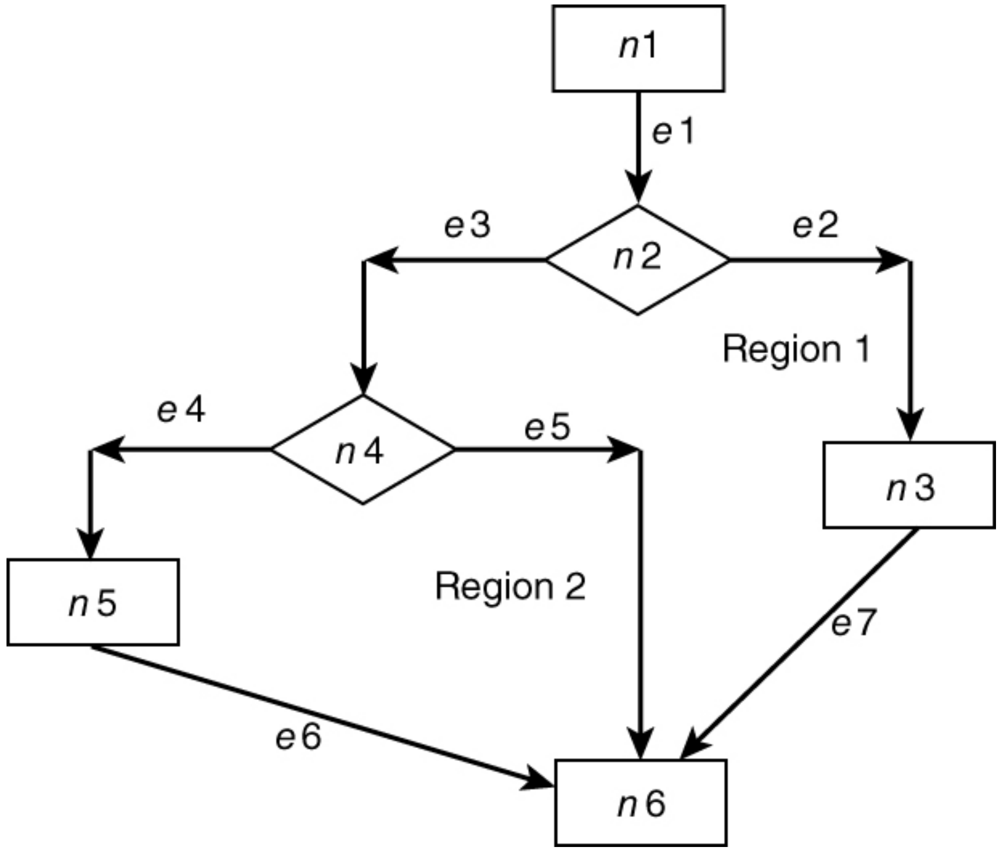
Halstead's Complexity Metric (cont.)

- Calculated values
  - Program vocabulary: $n = n1+n2$
  - Program length: $N = N1 + N2$
  - Program volume: $V = N*(\mathrm{Log}_2\, n)$
- Measures lexical complexity of program, not program structure or logic

McCabe's Cyclomatic Complexity

- Rationale: program quality is related to control flow complexity
- Based on program flow graph, calculate
  - E = number of edges in graph
  - N = number of nodes in graph
  - Cyclomatic number = E – N + 2
  - CN = number of closed regions + 1
  - = number of branching nodes + 1
- CN < 10, low risk, CN > 50, very high risk
- Note CN is also the number of linearly independent paths through the program

McCabe's Cyclomatic Complexity



- E = 7, N = 6, CN = 3

Henry-Kafura Information Flow

- Measures information flow between modules
  - Parameters
  - Global variable access
  - Inputs
  - Outputs
- Doesn’t consider complexity of modules themselves
- Fan-in = number of incoming pieces of information
- Fan-out = number of outgoing pieces of information

Henry-Kafura Information Flow (cont.)

- Complexity of a module is
  - (Fan-in * Fan-out) ^ 2
- Complexity of a system is the sum of the complexity of all the modules
- Later modified by Henry and Selig to include internal complexity of module
  - C_i * (Fan-in * Fan-out) ^ 2
- Can also view Fan-in as number of modules that invoke that module, plus number of global variables accessed
  - Similar for Fan-out

Card-Glass Complexity

- Three measures using Fan-in and Fan-out
  - Structural complexity of a module
    - $S_m = (\text{Fan-out}_m) \wedge 2$
  - Data complexity of a module
    - Let P_m = number of variables passed to and from the module
    - $D_m = P_m / (\text{Fan-out}_m + 1)$
  - System complexity
    - $C_m = S_m + D_m$

OO Complexity

- Good OO systems preserve OO properties
  - Abstraction
  - Reuse
- Set of metrics identified by Chidamber and Kemerer in 1994 (C-K metrics)
  - Weighted methods per class (WMC)
  - Depth of inheritance tree (DIT)
  - Number of children (NOC)
  - Coupling between object classes (CBO)
  - Response for a class (RFC)
  - Lack of cohesion in methods (LCOM)

OO Complexity (cont.)

- WMC
  - Apply favorite complexity measure to each method in a class and add
    - Just count methods??
  - Higher number indicates more complexity and more difficulty to modify and test class
- DIT
  - Maximum path length from leaf to root in tree
  - Higher number implies more inheritance
    - More reuse (good)
    - More complexity (bad)

OO Complexity (cont.)

- NOC
  - Number of subclasses of a superclass
  - Higher number implies
    - More reuse (good)
    - Greater dilution of the abstraction represented by the superclass (bad)
- CBO
  - Essentially same as traditional coupling (more later)
- RFC
  - Number of methods used to respond to a message

OO Complexity (cont.)

- LCOM
  - Pairwise compare all methods in a class
  - P is the set of pairs that have no common instance variables
  - Q is the set of pairs that have one or more instance variables in common
  - LCOM = $|P| - |Q|$ if $|P| > |Q|$
  - LCOM = 0 otherwise

Coupling and Cohesion

- Attempt to measure the “easy tos” of a system
- **Cohesio**n measures the intraconnectivity of a module
  - Highly cohesive modules should do one task
  - Want **high (or strong) cohesion**
  - Important because, when making changes, all relevant information is in one place and there’s no distracting information
  - Good cohesion is hard to do

Coupling and Cohesion (cont)

- Levels of cohesion (listed strong to weak)
  - Functional: clearly does one thing
  - Sequential: one thing, but “less clearly”
  - Communicational: tasks rely on one common shared data structure
  - Procedural: tasks that occur in order are in same module
  - Temporal: tasks that occur near each other in time are in same module
  - Logical: tasks that do same kind of thing are in one module
  - Coincidental: tasks are in same module with no underlying reason

Coupling and Cohesion (cont.)

- **Coupling** measures the interconnectivity or interdependency of modules
  – Not the number of connections, but the kind
  – Want **low (or loose) coupling**
  – Important because changes are less likely to affect multiple modules when there’s low coupling

Coupling and Cohesion (cont.)

- Levels of coupling (listed from tight to loose)
  - Content: modules access each other’s data directly
  - Common: shared access via common global data structure
  - External: ties to input/output formats or devices
  - Control: some ”control flag” tells called module what to do
  - Stamp: portions or representations of data structures are passed as arguments
  - Data: argument list of atomic values from calling module
  - None: ideal but impossible

Coupling and Cohesion (cont.)