Acyclic complexity

Jannis Bloemendal
Dipl. Inform. (FH)
Communicatie Media Informatica
University of Applied Sciences Rotterdam
Email: bljan@hr.nl

Abstract. In this paper we propose a linear code complexity measure from the perspective of testing based on isolated condition paths. The new measure covers conditions, nested and repeating structures.

1 Introduction

A Progam consist of a sequences of instructions. The instructions can be categorized into syntax, logic and arithmetic determining the program control flow.

Software complexity is related with modularity, coupling and cohesion hence quality. 40% to 80% of software costs are traced on maintenance and approximatly 40% on fixing defects [SG11].

Analysing code complexity helps to identify risk, finds potential defects to test critical functionality in detail, increase quality, cohesion and decrease maintainance [RB11].

2 Existing measures

Cyclomatic complexity (CC) has been widely discussed by various authors. The most used metric has been formulated by Thomas J. McCabe in 1976 [McC76].

$$m = e - n + 2 \tag{1}$$

Where, e = the amount of edges. n = the amount of nodes.

CC is linear in it's nature and correlation with lines of code (LOC), which is why Graylin JAY et al. [al.09] are suggesting to implement LOC as complexity measure.

Mir Muhammd Suleman Sarwar et al. [MMSS13] are pointing out that it neither takes into account the difference between combined decisions, elementary conditions nor repeating structures. Their adaptation of CC includes loop iterations

$$V(G)* = V(G) + \prod_{i=1}^{n} P_i$$

$$P_i = U_i - L_i + 1$$

Where, P_i = No. of iterations of ith loop. U_i = upper bound of ith loop. L_i = lower bound of ith loop and $V(G)*$ = adjusted cyclomatic complexity for any control flow graph "G". The measure combines control flow and statements exercised.

2.1 Coverage

Test coverage metrics are providing quantitative representation of tested structures. The approach is to maximize coverage while minimizing testing effort. Each condition branches the flow into two control sub-paths, and determines the progression of the programm. The path of an isolated decision d_j thereby is given due to decisions and conditions (MC/DC) [RB11]. Let's define a condition c_j and it's path $\gamma : C \times \Sigma \to C^*$

$$\Sigma = \{true, false\}$$

$$C = \{c_0, c_1, c_2, c_3, ..., c_n\}, c_i \mapsto \Sigma$$

$$\varepsilon : C \to \Sigma \times C$$

$$D \subseteq \{C^*, S, E\}; d_i \in D$$

The transition function $alpha$

$$\alpha : D \times \Sigma^* \to \Sigma \times D$$

and the transition $\vdash$

$$\vdash \subseteq (\Sigma^* \times D \times \Sigma) \times (\Sigma^* \times D \times \Sigma)$$

The possible combinations of conditions on γ are arising through c_j's preceeding and succeeding condition flow. An acyclic transition path through c_j is described due to passing each condition once

$$\begin{aligned} P_j &= (v_i, d_i, b_i), ... \vdash (v_j, d_j, b_j) \vdash^* (v_n, d_n, b_n) \\ &= (c_i, b_i), ... \vdash (c_j, b_j) \vdash^* (c_n, b_n) \end{aligned}$$

where

$$\alpha(d_n, v_n) \mapsto E$$

with the mantle

$$P = (b_i, v_i, d_i) \vdash^* (b_n, v_n, d_n)$$

Listing 1. Indipendent structure

```
if (a>5) {
    ...
}
if (b<3) {
    ...
}
if (c<11) {
    ...
}
```

Listing 2. Dependent structure

```
if (i<11) {
    if (j<3) {
        ...
    }
    if (k>7) {
        ...
    }
    ...
} else {
    if (1>5) {
        ...
    }
}
```

Table 1. Complexity of possible condition path structures

$\|C\|$	LOC		CC	iso. paths		AC	
	min	max		max	min	max	min
1	1	l_0	1	2	2	1	1
2	1	l_1	2	4	3	3	2
3	1	l_2	3	6	4	6	3
4	1	l_3	4	8	5	10	4
5	1	l_4	5	10	6	15	5

3 Acyclic complexity

Instead of combining arithmetic and logic we propose a metric reflecting the logical test effort indicated by the amount of paths. The concept of basic paths described by [McC76] neglects loops and nesting. 100% sub-path combination coverage is intricate due to exponential effort. Conditions are providing the logical structure of programs (Listing 1, 2). Loops and recursions are construed due to their invariant and boundary [RB11], which is why their quantitative iteration increases inclination not path length. Let's define the inductive start of independent

$$\begin{aligned} \alpha_i = &(c_0, true) \to (c_1), (c_1, false) \to (c_1), \\ &(c_1, true) \to (E), (c_1, false) \to (E) \end{aligned}$$

and dependent structures

$$\begin{aligned} \alpha_d = &(c_0, true) \to (E), (c_0, false) \to (c_1), \\ &(c_1, true) \to (E), (c_1, false) \to (E) \end{aligned}$$

The isolated condition paths are delineate $log(2^n)$ subset of all transition combinations from beginning to end. If we draw the spanning trees the first structure consists of four paths, where the second consists of three complete isolated paths.

The nesting of loops and conditions doesn't increase testing effort from the perspective of parameter combinations, the amount of paths yet decreases reciprocal with nesting. The perceived complexity of dependent structures increases contrary due to remembering previous preceeding conditions (Table 1).

When every condition is isolated the amount of condensed edges in the condition flow matrix represents an intuitive measure indicating the complexity of the program flow (Table 1, 2, 3).

The lower and upper bounds of possible isolated condition transition paths are

$$
\begin{gathered}
2 * |C| <= |T| <= |C| + 1 \\
C_n \subset C; \quad m_u = |C| + |C_n|
\end{gathered} \tag{2}
$$

with $C_n \subset C$ nested conditions. The metric m_u is undifferentiated regarding condition nesting level. The subjective perception of increased complexity due to nesting and combinations isn't taken care of, which is why we define the condition nesting function $\lambda(c_i)$

$$
\begin{gathered}
n = |C| \\
\frac{n(n+1)}{2} <= m_{acyclic} <= n \\
m_{acyclic} = n + \sum \lambda(c_i)
\end{gathered} \tag{3}
$$

4 Conclusion

The proposed acyclic linear-quadratic complexity measure (3) reflects the logical test effort of isolated condition paths. It provides a reasonable value hindsight logic, repetition and structure (nesting) and an intuitive quantitative value according to subjective perception. It correlates better with the perceived and logical complexity.

Table 2. α_i Flow Matrix

	s	c_0	c_1	e
s		t		
c_0			(t,f)	
c_1				(t,f)
e				

Table 3. α_d Flow Matrix

	s	c_0	c_1	e
s		t		
c_0			f	t
c_1				(t,f)
e				

References

[al.09] AL., Graylin J.: Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship. In: . *Software Engineering & Applications* 2 (2009), June, S. 137–143

[McC76] MCCABE, Thomas J.: A complexity measure. In: *IEEE* (1976), Dec, Nr. 4, S. 308–320

[MMSS13] MIR MUHAMMD SULEMAN SARWAR, Ibrar A. Sara Shahzad S. Sara Shahzad: Cyclomatic Complexity. In: *IEEE* 1 (2013), Jan, Nr. 5, S. 274–279

[RB11] REX BLACK, Jamie M.: *Advanced Software Testing Vol. 3*. Santa Barbara, CA : rookynook, 2011

[SG11] SOUMIN GHOSH, Prof. (Dr.) Ajay R. Sanjay Kumar Dubey D. Sanjay Kumar Dubey: Comparative Study of the Factors that Affect Maintainability. In: *International Journal on Computer Science and Engineering (IJCSE)* 3 (2011), Dec, Nr. 12, S. 3763–3769