Fancy title goes here

Zalán Szűgyi
Department of Programming Languages and Compilers,
Eötvös Loránd University
Pázmány Péter sétány 1/C H-1117 Budapest, Hungary
Email: lupin@elte.hu

Zoltán Porkoláb
Department of Programming Languages and Compilers,
Eötvös Loránd University
Pázmány Péter sétány 1/C H-1117 Budapest, Hungary
Email: gsd@elte.hu

***Abstract*—In software development testing plays the most important role to discover bugs and to verify that the product satisfies its requirements. Several methods exist to check code correctness. Less strict ones require fewer test cases and consume less resources, however they may discover fewer errors. Chosing test methods is always a compromise between the code correctness and the available resources. In this paper we would like to help on this choice. We analyse two important testing methods, the Decision Coverage and the more strict Modified Condition / Decision Coverage in several aspects. We discuss how these aspects are affected the difference of the necessary test cases for these testing methods. The analysis is done on open source programs written in C++.**

I. Introduction

... software testing [**?**] ... The main goals of code coverage analysis is to find areas of a program not exercised by a set of test cases, and create additional test cases to increase coverage, which is an indirect measure of code quality [**?**].

Code coverage analysis is a structural testing technique (white box testing), where it test program behavior is compared against the apparent intention of the source code. Different types of analysis require different sets of test cases:

a) Statement Coverage (SC): requires that each statement of a program must be invoked at least once. The main advantage of this method is that is can applied directly on object code. However this method is insensible to some control structures. See the code snippet below:

```
T* t = 0;
if (condition)
  t = new T();
t->method();
```

One test case – where the variable `condition` is true –, may provide a 100% statement coverage, becouse all the statements are invoked. In that case the program works properly and we may recognize it is faultless. However in real application the `condition` can be false, which might cause non-deterministic behavior or segmentation fault.

b) Decision Coverage (DC): enhances statement coverage by requiring that every decision must be evaluated both as true and as false. Thus the previous problem will be discovered in testing time. However this method ingores branches within boolean expressions, which occur due to short-circuit operators. Let consider the boolean expression $A||B$. Two test cases (where $A == true$, $B == false$, and $A == false$, $B == false$) can satisfy the requirement of DC, however the effect of B is not tested. Thus these test cases cannot distinguish between the decision $A||B$ and the decision A.

c) Condition / Decision Coverage (C/DC): requires that all the arguments in a logical expression must be evaluated both as true and as false. This method obviously solves the problem of DC, however it takes for a huge overhead due to the increase of arguments in the logical expression increases the number of required test cases exponentially.

d) Modified Condition / Decision Coverage (MC/DC): is derived from (C/DC) testing method, however it need less test cases to achieve 100 % coverage. This testing method has three requirement:

1) every statement must be invoked at least once,
2) every decision must be evaluated both as true and as false,
3) each condition must be shown to independently affect the outcome of the decision.

The independence requirement ensures that the effect of each condition is tested relative to the other conditions. The logical expression $A||B$ is fully covered with three test cases, where the arguments are $(false, false), (true, false), (false, true)$.

More information on these coverage methods and others can be found in [**?**], [**?**].

In this paper we concentrate on DC, and MC/DC testing methods, and examine how many test cases are neccessary to reach 100% coverage. It is clear that more test cases are needed to satisfy the requirements of MC/DC. But it is not so trivial how much can be spared when testing by DC instead of MC/DC. To answer that question we analyse several open source projects written in C++ programming language. We discouss our results in several aspects: McCabe metrics [**?**], nesting, and maximal argument number in decisions. We examined how these aspects affect the difference of the necessary test cases. [**?**]

Our paper is organized as follows: Section **??** and Section **??** describes the way how we computed the necessare test cases for DC and MCDC testing methods. These methods are applied on several open source projects and we detail our results in Section **??**. We discouss the related work in Section **??**, and we conclude our results in Section **??**.