Fault Density, Fault Depth and Fault Multiplicity: The Reward of Discernment

Besma Khaireddine
University of Tunis El Manar Tunis, Tunisia
khaireddine.besma@gmail.com

Aleksandr Zakharchenko, Ali Mili
NJIT Newark NJ USA
az68@njit.edu, mili@njit.edu

Abstract—Using a semantics-based definition of faults, we discuss three distinct but related concepts, and illustrate their differences by means of simple experiments: fault density (number of faults in a program); fault depth (the minimal number of fault removals needed to make a program correct); and fault multiplicity (the number of atomic changes needed to repair a single fault).

Index Terms—fault, elementary fault, fault removal, fault density, fault depth, fault multiplicity.

I. INTRODUCTION

In [2], [7] Laprie et al. define a fault as the adjudged or hypothesized cause of an error [2]. Also, the IEEE Standard IEEE Std 7-4.3.2-2003 [1] defines a software fault as "An incorrect step, process or data definition in a computer program". Neither of these definitions gives us any useful usable insight into what a fault is nor how to remove a fault: The IEEE definition does not even try; the definition of Laprie et al. [2] depends on the definition of an error, which in turn assumes that we have a specification of what is a correct state at each step of a computation, clearly an unrealistic assumption (in practice it is barely realistic to assume that we have a specification of the whole program, let alone a specification of every step of its execution).

Due to space restrictions, we keep our discussions informal, referring the interested reader to [4], [6]. We consider the program as a sequence of (syntactic) atoms [5] and we let a (syntactic) feature be one or more (not necessarily contiguous) syntactic atoms in the text of the program. We use the concept of relative correctness, i.e. the property of a program to be more-correct than another with respect to a specification, and we define a fault in program P with respect to specification R as a feature that admits a substitution that would make the program more-correct, and an elementary fault as a fault that is either a single atom or a set of atoms such that no subset thereof is a fault. Also, we let a fault removal in program P with respect to specification R be defined by the pair of features (f, f’) such that f is a fault in P and program P’ obtained from P by substituting f’ for f is strictly more-correct than P (i.e. P’ is more-correct than P but P is not more-correct than P’).

We define the fault density of a program as the number of elementary faults in a program, the fault depth of a program as the minimal number of elementary fault removals that are required to make the program (absolutely) correct, and the fault multiplicity of an elementary fault as the number of syntactic atoms that form the fault. In the next sections we discuss empirical experiments intended to elucidate the differences between these metrics.

II. FAULT DENSITY VS. FAULT DEPTH

We consider the tot-info component of the Siemens benchmark [3]; this component has 307 LOC and comes with a test data set of size 1052. We seed this program with 7 changes (faults?) that come with the benchmark, and we run an experiment intended to locate and repair these faults. We generate successive mutations of this program, and check the relations of strict relative correctness between them. The result is shown in Figure 1. Each node of this graph represents a mutant of tot-info and each arc represents a relative correctness relation. The fault density of a node is the number of arcs going up from that node; the fault depth of a node is the shortest distance from that node to the top of the graph. Space limitations preclude us from a detailed discussion, but the reader can see how density and depth take different values for each node, and evolve differently along each arc.

III. FAULT DEPTH VS. FAULT MUTIPlicity

We run the same experiment as above, with the replace component (563 LOC) of the Siemens benchmark, in which we seed six changes (faults?), which are provided as part of the benchmark. When a node in the graph is not absolutely correct and admits no mutant that is strictly more-correct, we apply double mutation; the assumption then is that we are looking at a fault with a higher multiplicity (than 1). The result is the graph in Figure 2; the two arcs at the top of the graph represent fault removals of multiplicity 2 (i.e. we had to perform two mutations before we could achieve correctness enhancement).

IV. CONCLUSION

This paper stems from two definitions, which we present informally as follows:

• A fault is an opportunity to enhance the correctness of an incorrect program.
• A fault removal is the act of transforming this opportunity into a reality.
This may sound counter-intuitive, as it presents faults in a positive light: faults as opportunities. But we point out that the real measure of faultiness is not fault density, but rather fault depth: For a given fault depth, the higher the fault density the better, since higher density means more opportunities to enhance correctness.

REFERENCES


