9-2016

Why Pairwise Testing Works So Well: A Possible Theoretical Explanation of an Empirical Phenomenon

Francisco Zapata
University of Texas at El Paso, fazg74@gmail.com

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-16-68
To appear in *Mathematical Structures and Modeling*, 2017, Vol. 41.

Why Pairwise Testing Works So Well: A Possible Theoretical Explanation of an Empirical Phenomenon

Francisco Zapata and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
fazg74@gmail.com, vladik@utep.edu

Abstract

Some software defects can be detected only if we consider all possible combinations of three, four, or more inputs. However, empirical data shows that the overwhelming majority of software defects are detected during pairwise testing, when we only test the software on combinations of pairs of different inputs. In this paper, we provide a possible theoretical explanation for the corresponding empirical data.

1 Pairwise Testing: Empirical Data

Need for software testing. At present, many processes depend on computer programs. In software dealing with health issues and with complex technology like flying a plane or operating a nuclear power station, a program mistake can be fatal and/or catastrophic. It is therefore important to make sure that the software function correctly.

In the ideal world, we should be able to *prove* program correctness, so that we will be 100% sure that the program produces the correct results. However, at present, such formal verification is only possible for rather simple programs. For most programs, the only way to make sure that the program works correctly is to test it with different inputs and different parameters.

Testing is not a perfect way to verify software. Most program have many possible inputs and many possible setting. For each of the inputs, we have a large number of possible values. It is therefore not feasible to test all possible combinations of inputs. So, we have to select which combinations we can feasibly test.

Empirical data helps to decrease number of tests. Since there is usually a large number of possible inputs, we cannot test all possible combinations of these inputs, so we have to limit the number of inputs whose combinations we test.

The simplest approach is to consider each input one by one, and for each input, to try different values of the corresponding quantity. This approach helps find some program defects, but it misses many defects that can be hidden behind if-statements with two or more conditions.

Let us present a simple example related to real-number computations following the IEEE 754 standard. A program may contain a division x/y of two real numbers x and y. If we fix a value $y \neq 0$ and consider all possible values x, the division will be performed perfectly well. Similarly, if we fix a value $x \neq 0$ and consider all possible values y, the division will be performed perfectly well. Even for $y = 0$, we will get a legitimate infinite value. However, if both x and y are equal to 0, then – unless we explicitly instructed a computer what to do in this case – we will have a problem, since $0/0$ is undefined.

A natural next step is to consider *pairwise testing* when we consider all possible pairs of inputs, and for each pair, test different combinations of the corresponding values. The following step is to consider all possible triples of inputs, etc.

At first glance, since, as we have mentioned, most program have a large number of inputs, we should need to consider all possible combinations of such inputs to find all the program's errors. Interestingly, however, in practice, most program errors can be detected already on the pairwise testing stage. Specifically, the empirical data (see, e.g., [1]) shows that 84% of the defects are detected in pairwise testing. Out of the remaining 16% of the defects:

- 11% are detected when we perform triple-wise testing, and
- 4% require trying combinations of 4 inputs.

Only the remaining (1%) of the defects require combinations of 5 or more inputs. In other words, the vast majority of defects can be found by using pairwise testing.

Open problem. To the best of our knowledge, there is no convincing theoretical explanation for the above statistics.

In this paper, we provide a possible theoretical explanation for the empirical data – and thus, for the empirical efficiency of pairwise testing. The existence of such a theoretical explanation increases our confidence that pairwise testing is indeed a reasonable testing strategy.

2 Towards an Explanation

How testing is done: a general description. In principle, even for a single real-valued variable, there is an astronomical number of possible values. It is therefore not practically possible to test the program on all these values.

Usually, there is a certain time allocated for testing, and we only perform as many tests as we can fit within this time.

Testing often starts with considering the values of one of the variables. This way, we can catch some defects. However, it is well understood that if we only change the value of *one* of the variables, then many defects will remain undetected. So, the next step is usually to perform pairwise testing, i.e., for different *pairs* of variables, to test the program on different combinations of their values.

It is also known that even pairwise testing leaves some defects undetected. So, if we want a reliable software, we test it also on combinations of triples of variables, etc. For each size of variables-to-change, be it 1 variable, 2 variables, 3 variables, or even more, potentially we can have many possible combinations, but for each size, we are limited by time. In the first approximation, it is reasonable to assume that each stage of this testing process takes approximately the same time T:

- first, we spend time T testing the software on changed values of one of the variables,
- then, we spend time T on pairwise testing,
- then, we (may) spend time T on triple-wise testing, etc.

From the general description of testing to numerical estimates. Let us see how the above general description of the testing process translates into probabilities of detecting a defect.

During each stage, we spend approximately the same time T on testing. Thus, on each stage, we perform approximately the same number of tests, and so, we have approximately the same probability of detecting any given defect on this stage. Let us denote this probability by p.

After the first stage, we detect a defect with this probability p. With the remaining probability $1-p$, the defect will be undetected.

If we started with the second stage, then we would be get the same probability $1-p$ of not detecting the defect on this stage. To estimate the probability that the defect will not be detected after two consequent stages, we can take into account that the tests performed on different stages are (or at least should be) independent. Thus, the probability that we did not detect a defect after the first two states is equal to the product of the two probabilities:

- the probability $1-p$ that this detect was not detected on the first stage, and
- the probability $1-p$ that this detect was not detected on the second stage.

This probability is therefore equal to $(1-p)^2$.

Similarly, the probability that the defect is not detected after three stages is equal to $(1-p)^3$, the probability that this defect will not be detected after

four stages is $(1-p)^4$, and, in general, the probability that the defect will not be detected in k stages is equal to $(1-p)^k$.

Let us compare these numerical predictions with the observed values. According to the above estimations, the probability that we *did not* detect a defect after the first two stages is equal to $(1-p)^2$. Thus, the probability that the defect *was* detected after the first two stages is equal to $1-(1-p)^2$. This probability applies to all the defects, so the proportion of defects that are detected after the first two stages is equal to the same number $1-(1-p)^2$. We know that empirically, this proportion is approximately equal to 84%, so $1-(1-p)^2 \approx 0.84$, hence $(1-p)^2 \approx 1-0.84 = 0.16$, and $1-p \approx \sqrt{0.16} = 0.4$.

The probability that the defects were not detected after *three* stages is equal to $(1-p)^3$. Thus, the proportion of the defects that were not detected after three stages is also equal to $(1-p)^3$. So, the proportion of the defects that were not detected after the two stages but that were detected after the third stage is equal to the difference $(1-p)^2-(1-p)^3$ between:

- the proportion $(1-p)^2$ of the defects that were not detected after the first two stages, and
- the proportion $(1-p)^3$ of the defects that were not detected after the first three stages.

For $1-p \approx 0.4$, this proportion is equal to $0.4^2-0.4^3 \approx 16\%-6\% = 10\%$, which is very close to the observed amount of 11%.

Similarly, the probability that the defects were not detected after *four* stages is equal to $(1-p)^4$. Thus, the proportion of the defects that were not detected after four stages is also equal to $(1-p)^4$. So, the proportion of the defects that were not detected after the two stages but that were detected after the third stage is equal to the difference $(1-p)^3-(1-p)^4$ between:

- the proportion $(1-p)^3$ of the defects that were not detected after the first three stages, and
- the proportion $(1-p)^4$ of the defects that were not detected after the first four stages.

For $1-p \approx 0.4$, this proportion is equal to $0.4^3-0.4^4 \approx 6.4\%-2.6\% = 3.8\%$, which is very close to the observed amount of 4%.

So, our model indeed provides a very accurate description of the empirical data.

Acknowledgments

This work was supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation.

References

[1] R. Black, *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional*, John Wiley & Sons, 2007.