Measuring Testing Complexity

Javier Ferrer*, Francisco Chicano† and Enrique Alba‡
University of Mlaga, Spain
*Email: ferrer@lcc.uma.es †Email: chicano@lcc.uma.es ‡Email: eat@lcc.uma.es

I. Introduction

Since the birth of Software Industry there is a special interest in measuring the effort in terms of time and cost that a task requires. Nowadays, since applications are essential for Industry, the software developers need to measure different kind of aspects: quality, cost, reliability, etc. Complexity is another important aspect in which software engineers are interested. But, what does we mean with "complexity" of a software piece? Basili [1] defines complexity as a measure of the resources used by a system while interacting with a piece of software to perform a given task. If the interacting system is a computer, then complexity is defined as the execution time and storage required to perform the computation described by the program. If the interacting system is a programmer then complexity is defined as the difficulty of performing tasks such as coding, debugging, testing or refactoring. In this abstract we propose a new meaning for complexity: the difficulty of testing a program.

Such a measure of complexity, which we call *Testing Complexity*, could be useful to decide which is the best way of generating a test suite for the piece of work. It could be useful, for example, to decide the parameters of an evolutionary test data generator [3] prior to its execution. Our new complexity measure predicts the behaviour of an automatic test data generator based on random testing [2]. The measure is based on a Markov model of the program that is used for estimating the probability of traversing the branches of the program. In order to confirm the quality of our new complexity measure, we performed an empirical study using a benchmark of 1800 test programs automatically generated.

II. Testing Complexity

According to our measure a program A is more complex than another one B if it is more difficult to test, that is, if the number of test cases that a random test case generator has to generate for obtaining full branch coverage in A is larger than the number of test cases generated for B. For this reason we want to define a complexity measure closely related to the branch coverage obtained by a random test data generator. The definition of this new measure lies on a Markov chain that represents the program under test and, in addition, it is an estimation of the number of test cases that must be generated to obtain full branch coverage of the program.

A Markov chain is a discrete random process with the property that the next state depends only on the current state. A discrete random process is a system which is in a certain state at each step, with the state changing randomly between steps. The conditional probability distribution of the next step depends only on the current state of the system, and not on the state of the system at previous steps:

$$P(X_{n+1}|X_n, X_{n-1}, ..., X_1,) = P(X_{n+1}|X_n) \qquad (1)$$

The previous probabilities are called *transition probabilities*. The set of all states and the transition probabilities completely characterize a Markov chain. Since the system changes randomly, it is generally impossible to predict the exact state of the system in the future. However, the statistical properties of the system's future can be predicted. In particular, it is possible to compute the frequency of appearance of the states in the Markov chain.

In our case the states of the Markov chain are the basic blocks of the program and the transition probabilities are the probabilities of jumping from one basic block to another one (traversing a branch). The Markov chain is thus statically built from a program computing for each basic block the probability of jumping to the next basic blocks (details below). Once we have built the matrix of transition probabilities, we add a fictional link between the last state and the first one. This is necessary to accomplish the requirements of a Markov chain (we need a cycle).

The transition probability of a branch is computed after the decision of the branch. We recursively define this probability as follows:

$$P(c1 \,\&\&\, c2) = p(c1) * p(c2) \qquad (2)$$

$$P(c1 \,||\, c2) = p(c1) + p(c2) - p(c1) * p(c2) \qquad (3)$$

$$P(\neg c1) = 1 - P(c1) \qquad (4)$$

where $c1$ and $c2$ are conditions.

In order to completely specify the transition probabilities we need to define their value in the base case of the recursion, that is, for atomic conditions. We establish a $1/2$ probability when the operands are ordering relational operators $(<, \leq, >, \geq)$ because if one generates two random numbers, the resulting probability of the condition to be true or false is $1/2$. The actual probability in a random situation is not always $1/2$, but it is reasonable to chose this value because, on average, it is the value with less distance to the actual probability. In the case of equalities and inequalities the probabilities are p and $1-p$, respectively, where p is a parameter of the measure and its value should be adjusted based on the experience. Satisfying an equality is, in general, a hard task and, thus, p should be close to zero. This parameter could be highly dependent on the data dependencies of the program. The quality of the complexity measure depends on a good election for p. We delay to future work the thorough analysis of this parameter.

Once we build the Markov chain associated to the program we are analyzing, we can compute the stationary probability of the basic blocks (the frequency of appearance of the basic blocks in a program execution) and the probability of traversing any branch in one execution of a program. We define the *Testing Complexity* as the average of the branch probabilities with a value lower or equal to $1/2$. If a program has a low value of testing complexity then a random test case generator requires a large number of test cases to obtain full branch coverage. Equally, a low value of testing complexity implies low branch coverage for a random test case generator generating a fixed number of test cases. The testing complexity is always between 0 and $1/2$.

III. Analysis of the Testing Complexity

In order to check if the testing complexity is a good estimation of the difficulty to test a program we perform an empirical study using the following methodology. First, we generated 1800 programs in an automatic way. Second, we computed the testing complexity of all the programs. Third, we used a random test case generator (RND) to find a test case suite for each program. We perform 30 independent runs of the generator, which generates 150,000 test cases in each run. Finally, we computed the Spearman's correlation coefficient ρ, between the testing complexity and the average branch coverage obtained by RND for each program.

In the benchmark of programs the correlation between the testing complexity and the branch coverage is 0.732. This value is higher (in absolute value) than the one between branch coverage and the nesting degree ($\rho = -0.589$), which is the most correlated static measure as far as we know. This result confirms that our complexity measure is better than the existing measures to estimate the difficulty to test a program. In Figure 1 we show the average coverage obtained by RND against the testing complexity for all the programs. In the figure the correlation can visually be appreciated.

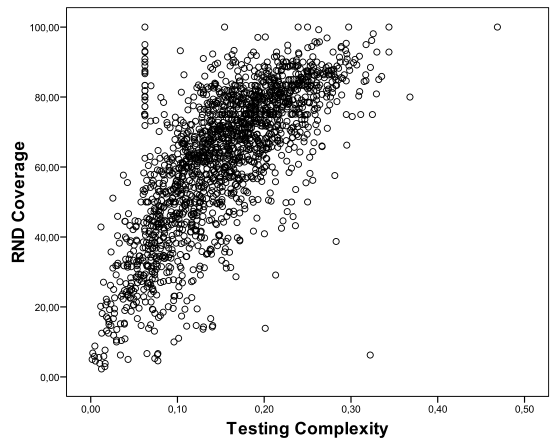

Fig. 1. RND average coverage against the testing complexity.

The Markov model of the program can be used not only to define the testing complexity, but to give an estimation of how many test cases RND has to generate to achieve a concrete branch coverage. As we explained in Section II, using the Markov model it is possible to compute the probability of a branch to be traversed during one single execution of the program. The inverse of this probability is an estimation of the number of test cases required by RND to cover the branch. If we compute these estimations for all the branches of the program and we sort them in ascending order the plot of the rank of the values against the value itself is a curve that represents the branch coverage against the number of generated test cases. This curve can also be statically computed for each program.

In Figure 2 we show this plot for a particular program together with the empirical plot obtained using the average coverage of the 30 independent executions of RND for that program. The resulting curves show that our theoretical model and the empirical data are very similar. The theoretical model is more optimistic because it does not take into account data dependencies. At the first steps of the algorithm, the empirical behaviour is better than the theoretical model, but when a high coverage is obtained (close to 90%), and less decisions have to be covered, the behaviour of the RND is worse than expected. One explanation for this behavior could be the presence of data dependencies in the program, which are not considered in the theoretical approach.

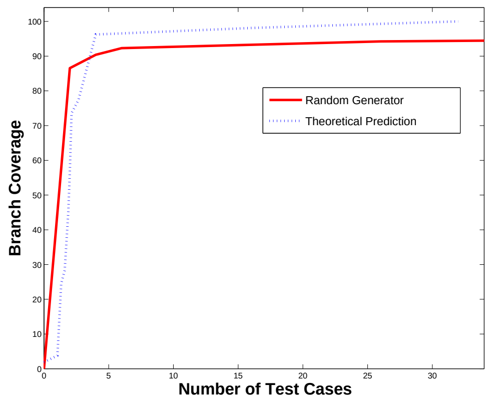

Fig. 2. Coverage against the number of test cases generated by RND and the theoretical prediction.

A. Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (the M* project). It has also been partially funded by the Andalusian Government under contract P07-TIC-03044 (DIRICOM project).

References

[1] V. Basili. Quantitative software complexity models: A panel summary. tutorial on models and methods for software management and engineering. *IEEE Computer society press*, 1980.

[2] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 213–223, New York, NY, USA, 2005. ACM Press.

[3] P. McMinn. Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*, 14(2):105–156, June 2004.