TEST AUTOMATION FOR QUALITY ASSURANCE: A RANDOM APPROACH

Paul Court and Omar Al-Azzam

Computer Science and Information Technology Department (CSIT),
Saint Cloud State University (SCSU), Saint Cloud, MN, USA

ABSTRACT

Testing is a necessary, but sometimes tedious chore for finding faults in software. Finding faults is essential for ensuring quality and reliability of software for industry. These are valuable traits consumers consider when investing capital and therefore essential to the reputation and financial well-being of a business. This research involves an ongoing trade-off between time and computational resources when testing via random selection of test cases versus complex logical means to find faults. More time will be devoted to an analysis of random test case selection and whether the amount of extra test cases run due to random selection is a viable alternative to the potential time spent fully evaluating the logic for coverage of a generic predicate. The reader will gain knowledge about the expectations for the increase in test cases if randomized selection is employed at some point in the process of testing.

KEYWORDS

Predicate testing, Fault detection, Simulation, Random selection, Logic coverage.

1. INTRODUCTION

Since the very first "bug" was found in a computer system and Grace Murry of Harvard University coined the phrase, fault detection in programming has been evolving [6]. The progression of software testing has been well chronicled and studied by some of the best minds in recent history. Its maturity can be correlated with the diversification in programming methodology [14]. There is no doubt that for essential, high-level functions, quality assurance is a must, but the dilemma arises when less critical software must be produced under the pressure of limited resources and tight time constraints, so efforts need to be focused on the critical risk areas [9]. In some cases, the logic necessary to cover all fault detection is extremely demanding. The correct balance between risk, cost, and quality assurance may be difficult to obtain. Randomness may play a role in test selection if the possible risks are known. There is a point where logical evaluation of predicates and clauses is so cumbersome that random test selection becomes viable as an option for a tester.

To be clear, a tester must have access/knowledge of the developer's decision-making logic and be able to translate the code of a predicate into Boolean statements suitable for their evaluation using advanced logical coverage techniques. This research will focus on one of the more powerful logic coverages, Restrictive Active Clause Coverage (RACC) [8]. Simulation will demonstrate how random selection of test cases compares to common quality assurance methods as well as weigh the pros and cons of partial uses of logical approaches and random chance both economically and for fault detection [21, 22, 24].

David C. Wyld et al. (Eds): FCST, CMIT, SE, SIPM, SAIM, SNLP - 2022
pp. 35-47, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.120804

2. BACKGROUND

When software is designed, input is necessary for the execution of the logic in a program. The potential values the input can take needs to be evaluated. The domain of these values can be partitioned into ranges that will determine the execution necessary blocks of code. Each time an input domain is partitioned, thought should be given to the input values to better understand what issues the implementation may have with the values entered [1]. These partitions evolve into Boolean values of true or false called clauses.

For example, a software may need to execute a block of code if an integer is between a minimum (x) and a maximum (y). The input domain includes the values of I that make the statement ($x < I < y$) either true or false. This can be considered a clause that has a Boolean value true or false. Clauses can be combined using logical operators to form more complicated structures called predicates, which determine code execution.

Testing predicates for proper performance uncovers potential faults in software. There are several logic coverages that testers employ to detect faults in predicates. One of the easier and weaker logic coverages is Predicate Coverage (PC). To achieve predicate coverage, any test case that has a predicate value of true can be selected along with any test case that evaluates the predicate to be false. Predicate coverage can usually be achieved with two test cases [1]. The most powerful logic coverage is Combinatorial Coverage. It occurs when every possible truth value combination is explored for each clause in a predicate. This is essentially exhaustive coverage and is achieved with $2n$ test cases, where n is the number of unique clauses. These are the extremes of logic testing. Active clause coverages such as RACC, are proven to detect faults efficiently, while minimizing the number of tests necessary [8]. To achieve RACC, each major clause must be evaluated to true or false as the minor clauses are held constant. Not an easily accomplished assessment for complex predicates. It is sometimes impossible to achieve.

Fewer faults in released software is the goal of testing and a measure of customer satisfaction, but so is reducing cost [16]. Logical evaluation of predicates can be time consuming and expensive. There is merit to using randomization to help with test case selection.
This research will consider random selection analysis for logic coverages discussed above.

H0: Randomizing test selection yields no advantage in achieving logic coverage in testing. Running unnecessary tests is expensive and consumes resources. Random selection of test cases is not a viable strategy in testing.

H1: A balance between randomness and intentional strategy will be the best method testers can employ to find a high degree of faults when logic testing becomes difficult or impossible under tight timelines.

Keeping in mind that adding value to the software while maximizing return on investment is the goal, we will show that a combination of random chance and logic will be most effective.

3. LITERATURE REVIEW

Often in testing as with businesses and industries, decisions can be automated and randomized. Data driven decision making has proliferated recently [18]. Predictive analytics use a combination of artificial intelligence, machine learning, statistical algorithms, data mining, and modelling to drive decision making and automation [2, 12]. Automation in testing has adopted these principles. Industry testing methods and tools are constantly under scrutiny for

improvement. The demand for better, faster, more effective designs can lead to better return on investment [11]. Test automation coupled with regression analysis and predictive modelling can lead to more effective test sets [23]. Tests that need to be repeated often or necessitate varied inputs work well with automated test framework (ATF) formats such as Selenium or Robot Framework. Random selection of inputs from a partitioned input domain can yield diverse and effective test cases. However, if test case selection is randomized, their selection should be prioritized. Some test cases available to a tester will be more involved in fault detection than others [5]. Therefore, random selection of test cases should be evaluated with due diligence. Unfortunately, automated, and randomized testing practices usually culminate with testers evaluating the processes to decide if fault detection via these methods accomplishes its goal.

4. METHODOLOGY

Deciding the input domain for testing will help define boundaries for possible variable values to determine clause Boolean. If input associations are established, a systematic method for random assignment can be automated. Rather than having testers take the time decide static input values, parameters can be set for automation. This process can be controlled by the system and once the guidelines are set, the system dynamically decides the test values for the test set. Logic coverage for test cases in this research will focus on restrictive active clause coverage (RACC). The number of tests required for partial or complete logic coverage will be studied.

First, we will consider completely random selection of input values given the input domain, directly determining clause Boolean values. The clauses will be combined using logical operators to form predicate Boolean necessary for test cases. The test case selection will be completely randomized, with test case repetition prevented to avoid redundancy. RACC will be evaluated studying the number of tests required to achieve coverage. Second, the idea of influential clauses and test cases will be considered. A superficial logical evaluation can uncover an influential RACC pair of test cases, thus determining two of the test cases in the test set. An influential test case is one that is shared by other clauses, contributing to coverage for several clauses simultaneously. Simulation will be employed to determine the impact of removing the known pair from the total cases while randomizing the selection of the remaining test cases. The number of test cases necessary to achieve RACC will be determined, with the number of excess cases necessary as the metric of evaluation. Predicates with three, four, and five clauses will be analysed. Predictive modelling practices will be used to provide estimates for larger predicates. This will give a tester the information necessary to weigh the resource of time spent to analyse clauses logically against the inefficiency of extra tests required due to random selection.

A demonstration can be used to help explain these ideas more clearly. For example, to decide if an integer input into a method is prime or a perfect square and in a certain range, the logic necessary to ensure that the correct values are entered requires testing. If this method is to be logically tested using Restrictive Active Clause Coverage some time would have to be devoted to these analyses. To determine whether the input is appropriate, it is necessary to check to see if the value falls between the minimum and maximum and is an integer. Then the integer can be evaluated to determine if it is either prime or a perfect square. This decision-making process can be represented with a three-clause predicate $p = a \wedge (b \vee c)$ where the clauses are determined using these variables:

a: the input is valid (integer and in range)
b: the input is prime
c: the input is a perfect square
$\wedge$: the standard logical operator “and”

∨: the standard logical operator "or".

Table 1 shows the main tool in logic coverage analysis, the truth table. When truth values for each clause are evaluated logically using the predicate operations, the truth value for the predicate is recorded to assist with test design.

To achieve predicate coverage, the weakest form or logic coverage, any test case that has a predicate value of true can be selected along with any test case that evaluates the predicate to be false. For example, the test set {1, 4} would be sufficient.

Table 1. Truth table for the predicate: $p = a \wedge (b \vee c)$

Test	***a***	***b***	***c***	***p***
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

Several other combinations of tests could be used, but predicate coverage could be attained with two well-chosen tests. Combinatorial coverage would require 23 = 8 test cases.

For a restrictive active clause coverage (RACC), each major clause must be evaluated to true or false as the minor clauses are held constant, as mentioned earlier [1]. One example for a RACC scenario would be for test cases 1 and 5. To activate clause a, notice in test case 1 clauses b and c have the value true while the value of clause a is true in test 1 and false in test 5. The predicate values are opposing. These test cases are RACC partners. Similar pairs of test cases need to be found in the table with each clause isolated as the major clause. Each clause's active state must be considered and how its state affects the value of the predicate. To have complete restrictive active clause coverage, four strategically chosen tests must be run; either {2, 3, 4, 6} or {2, 3, 4, 7}. Note that test cases 2, 3, and 4 are more influential test cases with test case 6 and 7, while still vital, are less influential in the outcome. Note also that test cases 1 and 5 could play a role in RACC coverage, however if these cases are selected randomly, there will have to be at least six test cases in a viable RACC test set.

These evaluations are tedious and require accurate representations of the logic for a predicate. They need to be considered to create test sets that are comprehensive with the goal of finding the maximum number of faults present in a code segment. Imagine if there were several more clauses involved. The analyses can become burdensome, time-consuming, and counterproductive cost wise. If a tester is underqualified or unmotivated to perform these analyses or the logic criterion are so cumbersome that they cannot be efficiently evaluated, can random selection of tests be a viable alternative. What are the risks?

Using probability to evaluate the example above, randomly selected test cases for predicate coverage where one test needs to have a predicate value of true and one test must evaluate to false, there is a 0.375 (3 out of 8) probability of selecting a test case with a predicate value of true. If the test cases are selected at random, to have the minimum number of cases of two, the scenario becomes: the predicate true is selected first, followed by the false or the predicate false

test can be first followed by the true. The probability of predicate coverage happening in the first two random selections is 0.46875. This implies that in 0.53125 proportion of random cases, more than the logically evaluated minimum number of cases would be necessary. This leads to running extra tests. However, if the situation is analysed differently and three tests are run randomly, the only result that would NOT produce predicate coverage is when all three tests are true or all three tests are false. An extra randomly selected test case increases the probability of satisfying predicate coverage by adding one extra test case to 0.703. These calculations are done allowing the test cases to be independently selected. If repetition is not allowed (the trials become dependent), the results above become slightly better.

5. EXPERIMENTAL RESULTS

Using the predicate in the example given, for predicate coverage, random selection of test cases with no repetition of test cases chosen shows that, on average, 2.776 tests would be necessary (standard deviation of 0.976) with 90% of trials achieving predicate coverage with four tests chosen at random. Unfortunately, four tests double the number of tests necessary in a logical evaluation. Figure 1 displays the results of 10,000 randomly selected test cases without repetition of selection. The selection is discontinued when predicate coverage is achieved, and the number of test cases is counted.

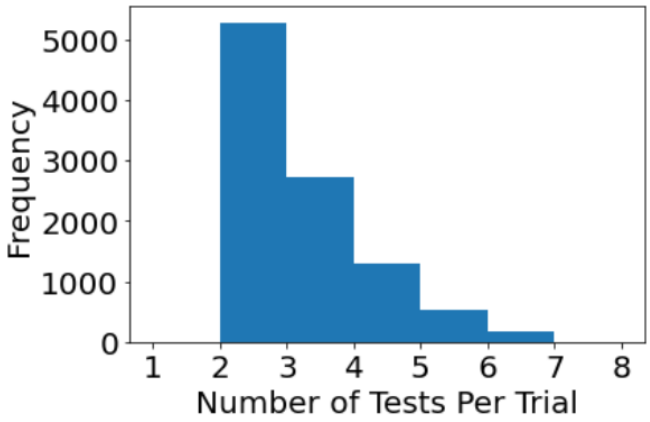

Figure 1. Predicate Coverage Random Simulation. The number of test cases necessary to achieve predicate coverage via random selection as demonstrated with 10,000 trials using the predicate in the example from Table 1.

In the case of the given example, a quick analysis of the logic would yield that test case 1 will have a predicate value of true. Another assumption would be that the predicate will be false more often than it is true. If we decide to choose test case 1 to be one of our predicate coverage test cases but let randomness decide the other, the total number of test cases necessary will obviously be larger than the minimum number of two and fewer than the four previously determined to be necessary for 90% fault detection by pure randomness. Figure 2 displays the results of 10,000 trials of this scenario. Test case 1 is assumed to be true and the other seven test cases are randomly selected. When a test case is found to have predicate value of false, the trial ceases and the number of extra cases is noted.

The average number of tests necessary under these conditions becomes 1.324 with standard deviation 0.553 tests, bringing the total number of tests necessary to 2.324, on average with 90% predicate coverage in 2 additional tests. A total of three tests will achieve predicate coverage for

the scenario described. When one test case is chosen logically and one is chosen at random, there is a gain of one test case from purely random selection.

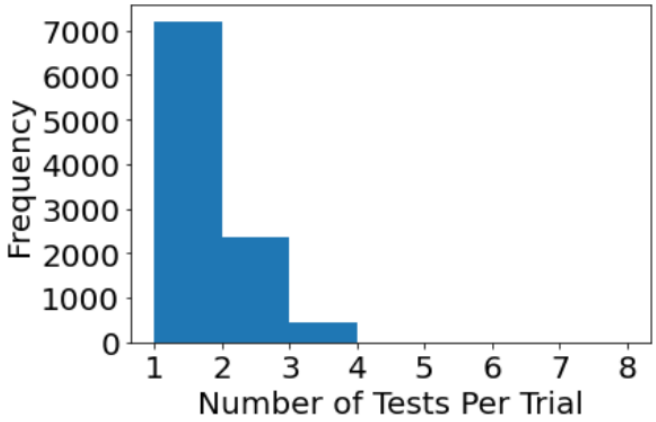

Figure 2. Predicate Coverage Simulated with One Test Case Removed. The results of 10,000 random trials evaluating predicate coverage with one test case determined logically and the other determined randomly.

This is very similar to a hypergeometric distribution (Equation 1) given that two tests are selected, one is required to have predicate value true and the other false. As the number of tests increases, the necessity is finding a predicate with opposing values to those already selected.

$$p(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

Equation 1: Hypergeometric probability where N is the population size, *K* is the number of successes in the population, *n* is the number of draws and *k* is the number of successes in the trial. There must be mutually exclusive categories and the trials have probabilities that are dependent according to the previous trial from a finite population.

Probability analysis of these events is extremely interesting but not the direction of this paper. Simulation and predictive modelling will be used instead.

Predicate coverage is not very powerful in finding faults. Our focus will be on the more effective Restrictive Active Clause Coverage. Purely random selection of RACC test cases shows that, on average 6.840 tests would be necessary to achieve RACC with 90% coverage at 8 tests or exhaustive testing. The results are depicted in Figure 3. As noted, a logical analysis indicates that four would suffice. This would, again double the number necessary. However, a superficial analysis of the predicate can yield knowledge that clause c largely determines the predicate.

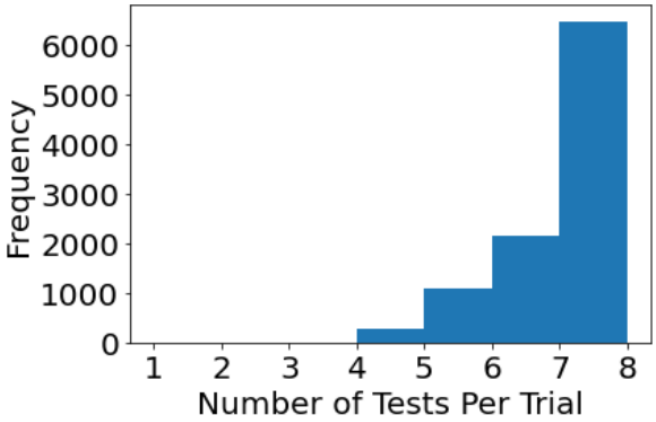

Figure 3. RACC Randomized. The results of 10,000 random trials to achieve restrictive active clause coverage in a three-clause predicate. The tests are randomly chosen until RACC is achieved.

If clause c is activated in test cases 3 and 4, these cases can be eliminated from the random selection and the process re-evaluated. Since RACC can be achieved with either of these minimum test cases {2, 3, 4, 6} or {2, 3, 4, 7}, test cases 2, 3, and 4 are higher priority with test cases 1, 5 helping to activate clause a, but not in the minimum test set. Test 8 is irrelevant in RACC coverage. If the logic to determine two high priority tests can be done easily, we can eliminate those tests from the random selection. Figure 4 shows a simulation of 10,000 trials randomly selected after test cases 3 and 4 were established to be part of the test set necessary to achieve RACC.

Other tests were randomly chosen until RACC was satisfied. The simulated results determined that an average of 3.964 extra tests need to be run, bringing the total number of tests in the test set to just under six. Coverage of 90% would take 6 more tests for a total of eight.

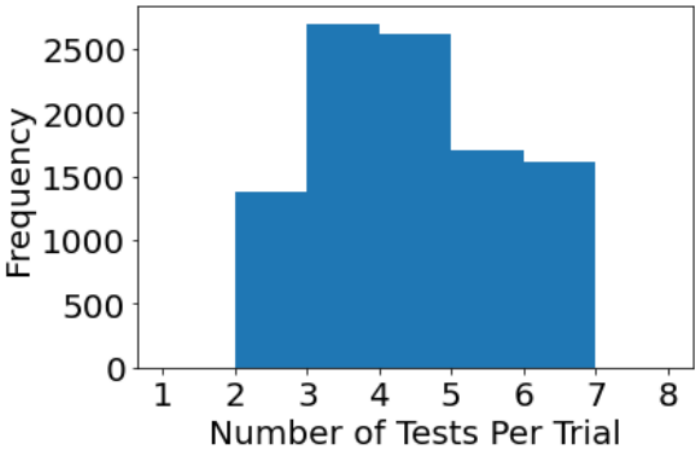

Figure 4. RACC Randomized with One Pair of Tests Removed. The results of 10,000 random trials to achieve restrictive active clause coverage for a three-clause predicate with an influential pair of tests removed and the remaining tests randomly chosen.

Further evaluating the scenario for many predicates involving three clauses it can be found that it is rare to find a predicate in which RACC is NOT achieved with four test cases.

Now consider a four-clause predicate such as p = (a ∨ (b ∧ c)) ∧ d. A truth table for this clause can be found in Table 2. Purely random test case selection to determine RACC will not be considered as there is little gain from the 2n = 16 total cases.

Table 2. A truth table for the four-clause predicate p = (a ∨ (b ∧ c)) ∧ d.

	a	b	c	d	p
1	T	T	T	T	T
2	T	T	T	F	F
3	T	T	F	T	T
4	T	T	F	F	F
5	T	F	T	T	T
6	T	F	T	F	F
7	T	F	F	T	T
8	T	F	F	F	F
9	F	T	T	T	T
10	F	T	T	F	F
11	F	T	F	T	F
12	F	T	F	F	F
13	F	F	T	T	F
14	F	F	T	F	F
15	F	F	F	T	F
16	F	F	F	F	F

To logically achieve RACC for this scenario, the minimum number of test cases would be five. A test set of either {3, 4, 9, 11, 13} or {3, 5, 6, 9, 11, 13} or {5, 6, 9, 11, 13} would provide RACC coverage.

If a RACC pair such as {9, 11} can be determined easily leaving the other three test cases to random selection, an average of 8.878 extra tests are needed. Bringing the total number of tests to about 11 instead of the necessary minimum of five. Figure 5 depicts the number of extra test cases randomly selected to achieve RACC for a four-clause predicate when an influential pair of tests is removed from the sixteen possible tests. The remaining tests are selected randomly until RACC is achieved.

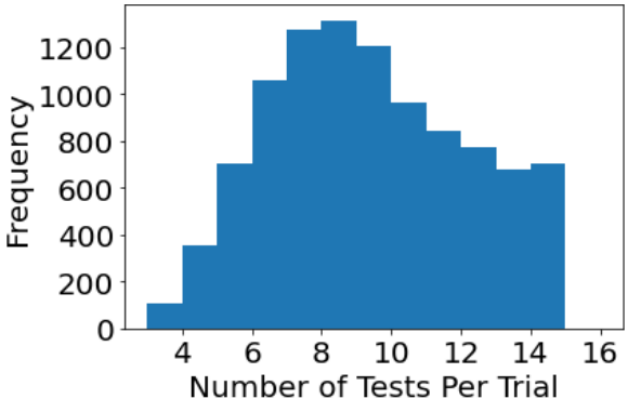

Figure 5. RACC Randomized with One Pair of Tests Removed. In a four-clause predicate with one fixed pair of RACC tests, the number of extra test cases necessary to achieve RACC via random selection of the remaining test cases.

Again, analysing many four-clause predicates, it is rare to find a predicate in which RACC is NOT achieved with five test cases.

Performing the same analyses on a five clause predicate the findings were that a minimum of six test cases were necessary to achieve RACC. If an influential RACC pair of tests are determined and the rest of the necessary case selections are left to chance, an average of 14.917 extra tests would be necessary for a total of 17 tests out of the possible 32. If RACC could be achieved with the minimum test cases of six, a tester would have to run 11 extra tests on average or almost double the number of tests.

Again, the probability analytics for this would be interesting and akin to the analysis of the game "Craps", however predictive modelling and simulation will provide our estimates for these predictions.

6. DISCUSSION

There is good evidence that RACC can be achieved by logically determining a minimum test set which if carefully selected can usually done with n + 1 test cases, where n is the number of unique clauses. It has also been shown that random test selection has little advantage over exhaustive testing of predicates. If a RACC pair of test cases can logically be determined, there becomes two fewer tests necessary to achieve coverage. Examining many predicates of with clause sizes 3, 4, and 5, a generic scenario was developed to determine test sets. Simulation of 10,000 trials for each clause size was run, showing the proportion of tests necessary to achieve RACC. The number of test cases necessary to achieve coverage was subtracted from the minimum necessary given the clause size and divided by the minimum, making a comparison possible. Figure 6 displays the results of the 30,000 simulations depicting the proportion of extra tests necessary.

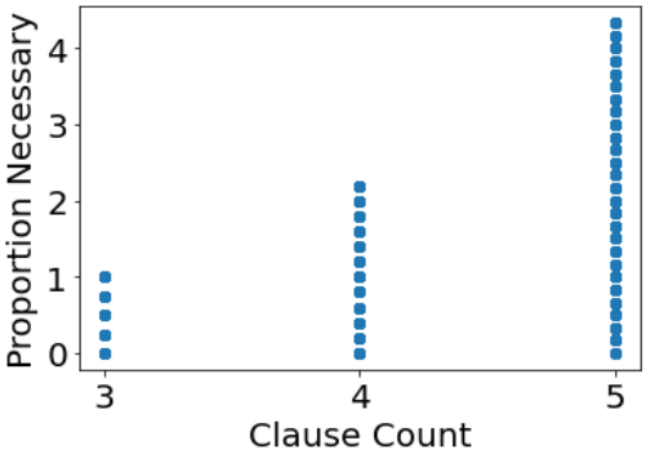

Figure 6. Proportion of Extra Tests per Clause. The proportion of extra test cases necessary to achieve RACC for predicates of clause size 3, 4, and 5 as determined by 30,000 simulated random test cases.

Linear, quadratic, cubic, quartic, and exponential regressions were run on the data to develop models to predict the performance of lager predicates under these conditions. There was little difference in the goodness of fit for the polynomial models. The linear model will not serve as a good choice, consistently under predicting the results. Similarly, the exponential model quickly overpredicting the proportion of test cases, making predictions higher than the total possible number of cases. Extrapolation of clause values outside the frame of reference studied can be

dangerous and after a clause size of eight, the models started to disagree. Figure 7 shows the linear, quadratic, cubic, and quartic models graphed over the simulated data. (Note: exponential regression is not pictured.)

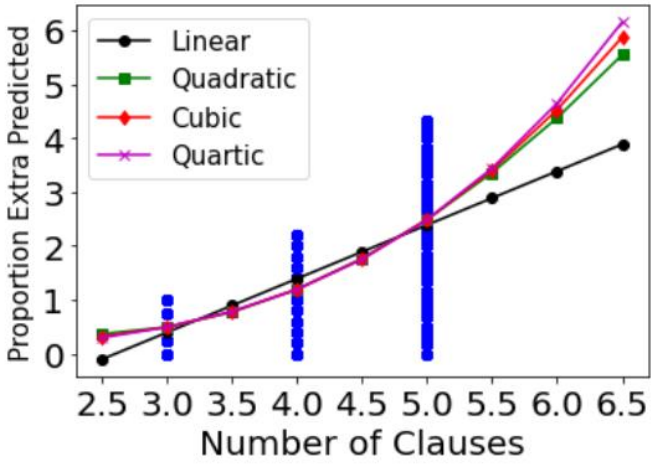

Figure 7. Predictions. Regression equations developed to help predict the proportion of extra test cases necessary to achieve RACC, using the number of clauses as the independent variable.

$$y = -2.600 + 0.996x \quad \ldots (2)$$
$$y = 2.070 - 1.441x + 0.304x^2 \quad \ldots (3)$$
$$y = 0.026 - 0.007x - 0.221x^2 + 0.513x^3 \quad \ldots (4)$$
$$y = 0.002 + 0.006x + 0.010x^2 - 0.0061 + 0.058x^4 \quad \ldots (5)$$
$$y = 10^{-4.148}(10^{1.0039})^x \quad \ldots (6)$$

Equations 2 – 6: Predictive equations for the proportion of extra tests run to achieve RACC for a predicate with *x* distinct clauses. Equation (2) is a linear model, equation (3) is a quadratic model, equation (4) is a cubic model, equation (5) is a quartic model, and equation (6) is an exponential model.

Table 3. Predictions for the proportion of extra tests necessary to achieve RACC given the number of distinct clauses in a predicate.

	6	7	8	9
Linear	3.38	4.37	5.37	6.37
Quadratic	4.39	6.91	10.04	13.78
Cubic	4.55	7.53	11.60	16.89
Quartic	4.68	8.10	13.15	20.27
Exponential	75.08	757.66	7645.62	77152.81

To summarize the findings from Table 3. It is possible to predict the average proportion of extra tests necessary to achieve RACC when two influential tests are removed from the random selection with the remaining tests necessary chosen by chance. Each predictive model was evaluated with the clause size six through nine entered as the independent variable. Table 4 summarizes the predictions in terms of raw number of extra tests given the number of clauses, n. The minimum logically evaluated number of test sets is n + 1. The maximum number of tests by exhaustion 2n. The predicted number of extra tests approximated by the models. The total number of tests run including the initial two chosen. The number of extra tests was computed, and the prediction is given as the next largest integer value. The prediction for quadratic, cubic,

and quartic were considered. Linear and exponential models were disregarded as under and overestimates, respectively. When the models disagreed, the most conservative (highest) predicted number of tests was considered.

Table 4. Summary of finding in terms of the average number of tests necessary to achieve RACC given the number of clauses.

n	$n + 1$	2^n	Prediction	Total
3	4	8	2	4
4	5	16	6	8
5	6	32	15	17
6	7	64	33	35
7	8	128	65	67
8	9	256	118	120
9	10	512	200	202

7. CONCLUSIONS

The null hypothesis that randomizing test selection yields no advantage in achieving logic coverage in testing was shown to be largely true. There is a slight advantage to randomization of testing, but it is difficult to determine the number of tests run randomly to achieve any specified degree of RACC coverage. Running unnecessary tests is expensive and consumes resources. A balance between randomness and intentional strategy will be the best method testers can employ to find a high degree of faults when logic testing becomes difficult or impossible under tight timelines. Using this predictive analysis may enlighten those to make informed decisions about the potential time necessary to evaluate a predicate logically and the time/expense of running extra test cases to detect the faults in a software. Understanding the risk consequences involved in is also a factor in the decision-making process. Also of note, it has been determined that very few predicates (0.65%) have four or more clauses [1]. However, if higher clause predicates need testing, the tester now can weigh the risks of random selection extra test cases against the time necessary evaluate the scenario logically.

Simulations for all trials employed python programming using Jupyter Notebook from Anaconda.Navigator. Random selection of test cases via input domain partitioning to determine truth values generated nonredundant test cases until the coverage criterion was met. The number of excess test cases necessary was determined. After much study of RACC coverage behaviours, a generic predicate was developed and used for simulations.

8. FUTURE WORKS

This analysis took into consideration that a tester would be able to determine an influential restrictive active clause coverage test pair with nominal time invested. More research would be necessary to determine the ramifications of selecting a test pair that was less influential. Another extension that would have merit for study would be incrementally increasing the test cases chosen logically to see the impact on the proportion of extra test cases necessary for coverage via random selection of the remaining test set. Similar ideas could be employed with different logic coverage choices, such as restrictive inactive clause coverage.

REFERENCES

[1] Ammann, Paul and Offutt, Jeff, *Introduction to Software Testing*, Second Edition, Cambridge University Press, 2016

[2] Chowdhury, Arnab Roy, (2020, March 21) *Test Analytics: What You Should Be Measuring in Your QA*, TestIM, https://www.testim.io/blog/test-analytics-qa/

[3] Clayton, Erna, (2020, January 27) *The Impact of Automated Software Testing on Native Manual Testing*, Developer Tip, Tricks & Resources, https://www.mabl.com/blog/machine-learning-in-testing-bots-vs-humans

[4] Elgabry, Omar, (March 17, 2017) *Software Engineering – Software Process and Software Process Models (part 2)*, Software Engineering — Software Process and Software Process Models (Part 2) | by Omar Elgabry | OmarElgabry's Blog | Medium

[5] Fang, C, Chen, Z, Xu, B. (2012) Comparing Logic Coverage Criteria on Test Case Prioritization, researchgate.net, https://www.researchgate.net/profile/Zhenyu-Chen-5/publication/257686390_Comparing_logic_coverage_criteria_on_test_case_prioritization/links/55d0150108ae6a881385e066/Comparing-logic-coverage-criteria-on-test-case-prioritization.pdf

[6] Hernandez, David Amrani (Dec 3, 2019) *History of Software Testing*, https://medium.com/@davidmoremad/history-of-software-testing-cfa461c4ae0a

[7] Hughs, Troy Martin, (2016), *SAS Data Analytic Development: Dimension of Software Quality*, John Wiley and Sons Inc.

[8] Kaminski, G, Ammann, P, Offutt, J. (2011), Better Predicate Testing, cs.gmu.edu, https://cs.gmu.edu/~offutt/rsrch/papers/ror-logic.pdf

[9] Kenett, Ron S, Fabrizio, Ruggeri, Faltin, Fredrick W, (2018) *Analytic Methods in Systems and Software Testing*, John Wiley and Sons Inc., First Edition.

[10] Kinsbruner, Eran, (2019, August 13) *Manual Testing vs. Automated Testing*, https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing

[11] Lee, Jihyun, Kang, Sungwon, Lee, Danhyung, () *A Survey on Software Testing Practices*, https://www.researchgate.net/profile/Sungwon-Kang/publication/260649940_Survey_on_software_testing_practices/links/54b7af070cf2e68eb2803d04/Survey-on-software-testing-practices.pdf

[12] Mackerras, Claire, (2020, January 2) *You Need Predictive Analytics for Your Software Testing: Here's Why, My Tech Decisions,* https://mytechdecisions.com/it-infrastructure/predictive-analytics-software-testing/#:~:text=Predictive%20analytics%20helps%20the%20testing,to%20drive%20better%20application%20efficiencies.

[13] Martinez-Fernandez, Silverio, *Continuously Assessing and Improving Software Quality with Software Analytics Tools: A Case Study*, IEEE Xplore, Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study - IEEE Journals & Magazine

[14] Nicola, (March 25, 2019), *Agile Testing vs. Waterfall Testing*, EuroSTAR, https://huddle.eurostarsoftwaretesting.com/agile-testing-vs-waterfall-testing/

[15] Nederkoorn, Cordny (2016, January). *Data Science from a Software Tester's Perspective*, Sweetcode, https://sweetcode.io/data-science-from-a-software-testers-perspective/

[16] Page, Alan, (), *The Cost of Software Testing?*, CIOReview, The Cost of Software Testing? (cioreview.com)

[17 faker] Sakinala, Krishna, (May 16, 2019) *Test Data Generation for Automation Testing*, Evoke Technologies, Test Data Generation for Automation Testing | Evoke Technlogies (evoketechnologies.com)

[18] Sarro, Federica (2018, May). *Predictive Analytics for Software Testing: Keynote Paper*, SBST'18, https://dl.acm.org/doi/abs/10.1145/3194718.3194730

[19] Scheier, Robert L., *How Predictive Analytics Will Disrupt Software Development*, TechBeacon, How predictive analytics will speed software development, improve quality (techbeacon.com)

[20] Sharma, Sudney, (2020, September 2) *Big Data – Testing Strategy*, https://www.loginradius.com/blog/async/big-data-testing-strategy/

[21] TestIM, (April 15, 2020) *Test Automation ROI: How to Quantify and Measure it.* TestIM, Test Automation ROI: How to Quantify and Measure It (testim.io)

[22] testIM Group (2020, March 21). *Test Analytics: What You Should be Measuring in Your QA*, testIM, https://www.testim.io/blog/test-analytics-qa/

[23] Vardhan, Harsh (2019, December 30). *Appling Data Analytics to Test Automation,* STICKYMINDS, https://www.stickyminds.com/article/applying-data-analytics-test-automation
[24] (December 27, 2011) *How to Calculate ROI for Test Automation*, TestingWhiz, How to Calculate ROI for Test Automation (testing-whiz.com)

AUTHORS

Dr Omar Al-Azzam is an Associate Professor of Software Engineering in the Department of Computer Science and Information Technology (CSIT) at Saint Cloud State University (SCSU). Dr Al-Azzam earned his BSc and MSc from Yarmouk University, Jordan and PhD from North Dakota State University (NDSU). Dr Al-Azzam main research interests are big data analytics, bioinformatics and data mining.

Paul Court is a graduate student in the Professional Science Master of Software Engineering (PSMSE) program at Saint Cloud State University (SCSU) in the Department of Computer Science and Information Technology (CSIT). Mr. Court earned a MEd in Mathematics from the University of Minnesota and a BA in Mathematics from the University of Minnesota, Morris.