New Software Testing Methodologies for Regulation Needs

1N.Sudheer, 2Dr Vidhushi Sharma
1Research Scholar, 2Professor
1,2Department of Computer Science & Engineering,
1,2Aligarh Muslim University, Aligarh, Uttar Pradesh, India
dr.vidhushisharma@rediffmail.com, nidamanuri.sudheer@gmail.com

ABSTRACT: Software Testing is modifying activity different software engineering projects, especially for large security models. The new tests cases can uses some hundred to several thousands. It significant computing locations and lengthy execution times. Software testing is finding errors, and maintenance overall software costs. The software development and testing methodologies, tools, and techniques have emerged over the last few decades promising to change software quality. We take a new prioritization model which prioritizes test cases in descending order for Component Based Software Development (CBSD) in the concept of Prim’s algorithm. Our main aim is to solve and take the model in test case prioritization algorithm take CIG (Component interaction Graph) as input for a medium/ large size CBSD (Component Based Software Development Process) in real-time system as an example and to generate prioritized test cases in descending order. We have used genetic algorithm to change the test case and applying conditional in source code. Test case data is generated automatically using genetic algorithm is change and output performs is generated by random testing. Software tends to get more powerful. A hardware, and new software, technology uses nonlinear rates with Moore's Law at the limit. Software test is systems is more and more services and features, Relatively fewer some new services and features are adequately tested. It represents assessment of head, shoulders, tail, and rump patterns did not provide additional information that could improve gender determination in nestlings at the time of banding. Gendre orientation task in view of shading estimation on computerized photographs of posteriors and tails did not enhance determination by eye, but rather shading estimation from an examined back end plume drew nearer 100% exactness. We give a discriminant capacity mathematical statement in view of red, green, and blue shine values (RGB) of a checked posterior plume and propose this as a proficient and powerful technique for sex determination in Lesser Kestrel settling.


1. INTRODUCTION
Software testing is as old as the slopes ever. The testing of software is an essential method for surveying the product to decide its quality. Since testing ordinarily devours 40 - half of improvement endeavors, and expends more exertion for frameworks that require more elevated amounts of unwavering quality, it is a noteworthy part of the product building. Current software frameworks must be to a great degree dependable and correct.[2] Automatic strategies for guaranteeing software accuracy range from static methods, for example, model checking or static investigation, to element systems, for example, testing. Every one of these methods have qualities and shortcomings: model checking is programmed, thorough, yet may experience the ill effects of adaptability issues. Static examination, then again, scales to expansive projects however may give an excessive number of spurious notices, while testing alone may miss vital mistakes, since it is innately fragmented. Testing, check, assessment and acceptance are fundamental parts of the development of atmosphere models. Nonetheless, these procedures are imbued into the way of life of displaying focuses and are frequently not particularly recognized.[1] We recognize acceptance and confirmation as in Post and Kendall [2] where approval alludes to examination with perceptions and check alludes to correlation with explanatory experiments and computational items.
Software issues: Inadequate software execution, Data seeks that yields off base results. Off base information alters and ineffectual information alters, Incorrect coding/execution of business guidelines, Incorrect calculation,[3] Incorrect information alters and insufficient information alters, Incorrect preparing of information relationship, Incorrect or deficient interfaces with different frameworks, Inadequate execution and security controls, Incorrect record taking care of, Inadequate backing of business needs, Unreliable results or execution, Confusing or deceiving information, Software ease of use by end clients and Obsolete Software, Inconsistent handling

A specific system way or to check consistence with a particular requirement[11] for which esteemed inputs dependably exist. By and by, the entire arrangement of experiments is considered as limitless, thusly hypothetically there are an excessive number of experiments notwithstanding for the least difficult projects. For this situation, testing could require months and months to execute. Along these lines, how to choose the most appropriate arrangement of experiments by and by, different strategies are utilized for that[4] and some of them are connected with danger investigation, while others with test building mastery. Testing is a movement performed for assessing software quality and for enhancing it. Thus, the objective of testing is systematical location of various classes of mistakes [12] in a base measure of time and with a base measure of exertion.

Figure 1: Test Information Flow
As there is rapid development in software industries there is a demand of increasing use of component based software development processes. A Component collaboration diagram delineates that interrelation among various segments. Consequently by speaking to segment collaboration utilizing CIG we can diagram the experiment sequences.[5] We will examine new experiment prioritization procedure which depicts part connection between various segments by giving CIG as information and taking into account it which can organize experiment arrangements in dropping request by applying Prim's calculation on it. In the branch scope we ensure that we execute each branch at any rate once For contingent branches, this implies, we execute the TRUE branch in any event once and the FALSE branch at any rate once conditions for restrictive branches can be compound boolean expressions a compound boolean expression comprises of a blend of boolean terms joined with coherent connectives AND, OR, and NOT Condition coverage.[6] In this paper we propose a model for programmed and upgraded experiment era. In our proposed technique the underlying experiment created utilizing restrictive scope to cover every one of the ways then hereditary calculation is utilized for improving the experiments. This is a productive methodology of upgrading experiment by utilizing both hereditary calculation and restrictive scope.

2. RELATED WORK
In today’s world day by day drastic changes are in software which needs to be more proficient for this component based software development process provides platform to detect faults in quick time and refine code shortly.[7] To perform such tasks developers/ testers needs to find out best one out of alternatives that is ‘next best’ known as ‘greedy’ method. Fundamentally eager techniques are basic, simple to-actualize and give answers for complex issues by choosing which next step will give the most effective result. Such calculations are called voracious on the grounds that ideal answer for each litter occasion will give a compelling the calculation doesn't consider the bigger issue in general. Covetous techniques works with two unique criteria sorted. Model-based testing is programmed era of proficient test methodology/ vectors utilizing models of framework prerequisites and indicated usefulness [24]. In this technique, test cases are determined in entire or to some degree from a model that portrays a few parts of the framework under test. These experiments are known as the unique test suite, and for their determination diverse methods have been utilized. On-interest access to a common pool of configurable figuring assets, for example, systems, servers, stockpiling, applications and administrations that can be quickly provisioned and discharged with insignificant administration exertion or administration supplier collaboration. Driving organizations, for example, IBM, Microsoft, Google, and Amazon have a personal stake in distributed computing [1]. In that capacity a few applications, stages, and frameworks are being produced to encourage the conveyance of figuring assets as administrations over the Internet [2] [3] [4]. It is expected that by 2016 more than 75% of Information Technology framework will be bought as an administration from administration suppliers [5]. In 2010, Garner evaluated that “the cloud administration business sector will reach $150.1 billion in 2014”. A late investigation of Market Research Media conjectures that U.S. government spending on distributed computing is entering an unstable development stage at around 40% CAGR throughout the following six years.

Figure 2: Top Application of Cloud

3. RESULTS AND DISCUSSION
A total of 82 research papers related to Testing as a Service were returned by the mapping study searching process. The research papers were then categorized and classified into 4 groups which are Cloud based Testing, Automated Test Case generation, Testing Frameworks and Cloud Application Development. The breakdown of research papers according to the four categories. Based the majority of the research papers focused on Cloud based Testing followed by Testing Frameworks Automated Test Case generation and Cloud Application Development contributed 12 papers. The number of research publications made per year for each paper category. Overall, research publications on Testing as a Service (TaaS) grew from two publications in 2010 to 31 publications in Dec 2015. Based research papers on Cloud based Testing and Testing Frameworks have the highest number of increase in publications as compared to other research paper category. [9] The publications were also classified based on different forum types, similar to systematic review. Cloud environments should be tested and measured for their performance, availability, security and scalability in order to support efficient delivery of services [25]. A majority of the research papers focused on Cloud based testing. From a total of the 82 papers, 38 papers discussed Cloud based testing. In order to analyze the test generation approaches.
4. RESEARCH METHOD

Test cases are created utilizing different test strategies to accomplish more successful testing. By this, product culmination is given and states of testing which get the best likelihood of discovering blunders are picked. Along these lines, analyzers don't figure which test cases to picked, and test strategies empower them to outline testing conditions in an efficient way.[10] Also, on the off chance that one joins a wide range of existing test systems, one will get better results rather in the event that one uses only one test procedure. Programming can be tried in two courses, in another words, one can recognize two distinct techniques:

1. Discovery Testing,
2. White box Testing.

White box testing is very successful in recognizing and determining issues, since bugs [12] can regularly be found before they cause inconvenience. We can quickly characterize this technique as testing programming with the information of the interior structure and coding inside the system [13] White box testing is likewise called white box investigation, clear box testing or clear box examination. It is a system for programming investigating [14] in which the analyzer has brilliant information of how the project segments cooperate. This technique can be utilized for Web administrations applications, and is once in a while down to earth for troubleshooting in expansive frameworks and systems [14]. Moreover, in [15], white box testing is considered as a security testing [6] technique that can be utilized to accept whether code execution takes after expected configuration, to approve actualized security usefulness, and to reveal exploitable vulnerabilities [15].

Discovery testing will be trying programming taking into account yield necessities and with no learning of the inner structure or coding in the system [16] In another words, a black box is any gadget whose workings are not comprehended by or available to its client in information transfers, it is a resistor associated with a telephone line that makes it inconceivable for the phone organization's gear to distinguish when a call has been replied. In information mining, a black box is a calculation that doesn't give a clarification of how it functions. In film making, a black box is a devoted equipment gadget: hardware that is particularly utilized for a specific capacity, however in the money related world, it is a mechanized exchanging framework that doesn't make its standards effectively accessible. As of late, the third testing strategy has been likewise considered [11] dark box testing. It is characterized as testing programming while as of now having some learning of its fundamental code or rationale [17] It depends on the inside information structures and calculations for planning the experiments more than discovery testing yet not as much as white box testing. This technique is essential when leading mix testing between two modules of code composed by two distinct designers, where just interfaces are uncovered for test. Additionally, this strategy can incorporate figuring out to decide limit values. Dim box testing is non-meddlesome and impartial in light of the fact that it doesn't require that the analyzer have admittance to the source code.[18]

Fluff Testing Summary: Random info Fuzz testing is regularly called fluffing Robustness Testing or negative testing. It is produced by Barton Miller at the University of Wisconsin in 1989. This strategy nourishes irregular data to application. The fundamental normal for fluff testing, as indicated by the [26] are: the info is irregular. The unwavering quality criteria: if the application. Crashes or hangs, the test is fizzled fluff testing can be computerized high degree. An instrument called fluff analyzer which shows reasons for established powerlessness, works best for issues that can bring about a project to crash, for example, cradle flood, cross-site scripting, foreswearing of administration assaults, group bug and SQL infusion. Fluffing is less compelling for spyware, some infections, worms, Trojans, and keyloggers. fuzzers are best when are utilized together with broad discovery testing techniques.[19]

UNIFIED MODELING LANGUAGE The unified modeling language or UML models have the same goal as any model but replace the graphical-style representation of state machines with the power of a
structured language. UML is to models what C or Pascal are to programs – a way of describing very complicated behavior. UML can also include other types of models within it, so that finite state machines and state charts can become components of the larger UML framework.

**UML state chart diagram** Model based test case prioritization using UML state chart diagram In this technique test cases are prioritized for component based system retesting. Here state chart diagram is considered for each component which represents state transitions among each component then it constructs Component Interaction graph (CIG) to show an interrelation within components.[20] Prioritization algorithm counts interrelation among components and interrelation within components. It depends on maximum state changes occur among and within components as well as database access during test case execution.

**MARKOV CHAINS:** Markov Chains are stochastic models. A specific class of Markov chains, the discrete-parameter, finite-state, time-homogenous, irreducible Markov chain, has been used to model the usage of software. They are structurally similar to finite state machines and can be thought of as probabilistic automata. Their primary worth has been, not only in generating tests, but also in gathering and analyzing failure data to estimate such measures as reliability and mean time to failure. [21] The body of literature on Markov chains in testing is substantial and not always easy reading. Work on testing particular systems and Work related to measurement and test can be found.

**Testing strategies** A test strategy is an outline that describes the testing approach of the software development cycle. A strategy for software testing integrates the design of software test cases into a well planned series of steps that result in successful development of the software [6][13]. Testing begins at component level and work outward towards the integration of entire computer based system. There are different types of testing strategies.[22]

1. Unit testing- it concentrate on each component function of the software as implemented in the source code. Components are then assembled and integrated
2. Integration testing- it focus on the design and construction of the software system. It also focuses on input and output and how well the component fit together and works together.
3. Validation testing- requirements are validated against the constructed software. It provides final assurance that the software meets all functional and performance requirement.
4. System testing- In this the software and other system elements are teste as a whole that overall system function and performance is achieved. In this paper we used unit testing we take source code and focus on the each component function of the source code and internal structure of the program’s source code.

![Figure 4: The Testing Strategy](image)

5. **Algorithm**

Majorly there are two different techniques; code based test case prioritization and model based test case prioritization. In code based method system is dependent on source code completely whereas in model based method system captures the structure and behavioral aspects. Also model based technique finds faults in very less time as compared to code based approach.[23]

**A. Kruskal’s Algorithm**

In this algorithm at each stage, the edge with the least cost is processed Assume G = (V, E) Keep track of connected components of graph with edges Initially components are single nodes At each stage, add the cheapest edge that connects two nodes not already connected Experimental study of Kruskal’s algorithm states that Kruskal can have better performance if the can be sorted in linear time; edges in each component are already sorted.

**B. Prim’s Algorithm**

There are quite limitations with kruskal’s method; so for our model based prioritization technique we finalized Prim’s method to prioritize test cases which are generated in random sequence which we have to arrange in some sorted order.

**Implementation Details of Prim’s Algorithm**

As discussed in model based prioritization testers need to check behavioral aspects of the system multiple state changes occurred, the test case is going to access the data base or single attribute or multiple attributes from a database access. Hence I am proposing new optimization technique Prim’s Algorithm to explore more effective prioritized test cases which can find out defects in earlier time and can maximize the
effectiveness of any medium/ large system. As already discussed Prim's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. [24] This means it finds a subset of the edges which converts it in a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it will only find a minimum spanning tree for one of the connected components. Framework of model based test prioritization using prim’s algorithm.

3. T1[intracount] = 0; T2[intercount] = 0;
4. If Ck (Si) -> Ck (Sj) Then w = 0. // If states are interacting in same component.
5. If Ck (Si) -> Cl(Sj) Then w = 1. // If states are interacting in different component.
6. Select any state of any component, set S = {s}.
8. intracount ++;
9. Find lightest weight interactive state such that one endpoint is in S and other is in V\S.
10. If (w = 0) Then
12. intracount ++;
13. Else
15. intercount++;
16. End If
17. If (V\S! = ø) Then Step 9. //If no more state connected
18. T = T + T1; //Add T1 into the T
19. T = T+ T2; //Add T2 into the T
20. Set T’ = Reverse of T.
21. Output T’

Prioritization using Prim’s Algorithm
In this module actual prioritization can be performed on the basis of CIG generated and number of test cases listed for every state with prior knowledge of Prim’s algorithm. In following table first column represents more weighted test cases i.e. DB direct which are prioritized first and second column shows test cases which have less priority.

<table>
<thead>
<tr>
<th>DBdirect (intrastate changes)</th>
<th>DBindirect ( interstate changes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>T2</td>
<td>T4</td>
</tr>
<tr>
<td>T3</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
</tr>
</tbody>
</table>

Prioritized test suites After sorting test cases in this module we are finally representing prioritized test cases in decreasing order according to its priority as shown below

Prioritized Test Case Sequence
Result set 2 of Prioritized Test Sequence
6. Proposed Algorithm

1. Make the control flow graph of the source code.
2. Apply conditional coverage on the control flow graph.
3. Generate initial test case randomly.
4. Generate a set of optimize test case using a genetic algorithm.

So in our methodology for producing optimize test case for testing process we make control flow graph for source code and generate test case randomly by applying conditional coverage on the control flow graph and then for producing optimize test case that can detect all the faults as much as possible we refined test case using genetic algorithm. Genetic algorithm produced a set of optimize test case that is used by the user. Here the user has authority to testing the source code by using test cases. Genetic algorithm has well defined steps such as initialization, selection, crossover and mutation. The crossover and mutation process produce new test cases that can detect maximum faults for the source code. So by using this methodology we can generate test case automatically that save effort, time and cost for testing process. The algorithm described above enhances the software testing strategies by saving effort and cost by generating test cases automatically.

7. Generating Tests

The difficulty of generating tests from a model depends on the nature of the model. Models that are useful for testing usually possess properties that make test generation effortless and, frequently, automatable. For some models, all that is required is to go through combinations of conditions described in the model, requiring simple knowledge of combinatorics. In the case of finite state machines, it is as simple as implementing an algorithm that randomly traverses the state transition diagram. The sequences of arc labels along the generated paths are, by definition, tests. For example, in the state transition diagram below, the sequence of inputs “a, b, d, e, f, i, j, k” qualifies as a test of the represented system.

Running the Tests

Although, tests can be run as soon as they are created, it is typical that tests are run after a complete suite that meets certain adequacy criteria is generated. First, test scripts are written to simulate the application of inputs by their respective users. Next, the test suite can be easily translated into an executable test script. Alternatively, we can have the test generator produce the test script directly by annotating the arcs with simulation procedures calls. While writing the automation code, adherence to good engineering practices is required. Scripts are bound to interact with each other and evolve as the software evolves. Scripts can be used for as long as the software is being tested so it worthwhile investing some time in writing good, efficient ones. With model-based testing, the number of simulation routines is in the order of the number of inputs, so they are generally not too time-consuming to write.
Collecting Results:
Evaluating test results is perhaps the most difficult testing process. Testers must determine whether the software generated the correct output given the sequence of test inputs applied to it. In practice, this means verifying screen output, verifying the values of internally stored data and establishing that space and time requirements were met.[12] MBT does not ease this situation. Output and internal variables must still be checked against the specification for correctness. However, MBT does add one new dimension that is very useful in practice: the verification of state. States are abstractions of internal data and as such can often be more easily verified. For example, the model itself will catalog each state change that occurs (or should occur) in the software as test inputs are being applied. Thus, the model acts as a very precise and detailed specification, informing the tester what inputs should be available and the values of each data abstraction that comprises the state of the software under test.

Figure 8. Some Model-based Testing Activities

An Executable Software Test
The tester specifies what each test case will be, what steps it will take and what data it will use at runtime. Word was chosen because an effective test spec has two parts, an informal, free-form section that tells the reader something about the system under test and the rationale for the tests outlined in the subsequent structured section of the spec containing the actual tests to run.[21] A word processor like Word is the natural choice for writing free form content. Writing a user interface for Word enabled testers to create the structured document section, as well. Each step in the workflow generates a very specific form of XML. The Intermediate Markup Language (IML) is the XML schema generated first. Its role is to faithfully represent the contents of the Word document so that the system can generate a web form version of the spec. The Failure Investigator component of the system uses this machine-readable version of the test spec to display the section of the spec related to a failed test. This ability of infrastructure to reference the spec speeds up the bug resolution process. The transformation that generates the second XML schema, XIML, is the process that generates test cases from the description given by the tester in the spec. Depending on the data provided by the tester, a spec with only a few Vars in it can expand to hundreds of test cases. The key thing to remember is that a test consists of two things: actions to take and data to use.[16] A single var is defined by the actions it will take; and the multiplicity of generated test cases is a function of the diversity of data these actions consume. Our executable software test specification system fully exploits this duality of executable test specs. The ultimate goal of all this xml processing is to get the data in a shape that the test execution runtime can consume. The final step in the workflow transforms the XIML into an XML format defined by the developer of the test execution runtime.[19].

Figure 9: Executable Software Test Specification Workflow

8. Verification in Climate Model Development
Test-Driven Development (TDD) Test-driven development (TOO) is a major element of agile development processes and consists of a short development cycle that alternates between development of tests and development of code which enables the tests to pass. First, the developer creates or extends a test which then fails because the necessary functionality has not yet been created.[5] Then the developer produces code just sufficient to pass the test. Finally, the
developer cleans up with an emphasis on removing any incidental redundancy. The process encourages progress in the form of rapid, incremental steps and is made practical through unit testing frameworks. To better understand the TOO workflow and its application in the context of numerical software, consider the process of developing a simple 10 linear interpolation procedure. The first step is to create a test. Usually the first test is extremely simple and is primarily intended to specify the desired interface for the procedure to be implemented.

```python
assert(0 == interpolate(x=[1,2],y=[0,0],at_x=1)
```

With this test in place, we begin to implement the interpolate procedure itself, but only so far as to enable successful compilation and execution of the test. A trivial implementation that returns zero will suffice. For the next test we choose data that have a simple linear relationship:

```python
assert(1 == interpolate(x=[0,2],y=[0,2],at_x=1)
```

Additional tests could then be to check that the behavior is correct when there are multiple intervals in the data, or when the interpolation is outside the domain, the data are degenerate, etc. After each test, the interpolate procedure is extended to ensure that all tests pass. For many it will be initially counterintuitive to proceed in such minute steps, but with practice these steps happen very quickly and lead to steady predictable development of complex features[16].

9. CONCLUSION AND FUTURE WORK
The systematic mapping process was described in terms of the research questions defined, searching keywords used, the exclusion and inclusion criteria. The results of the study was classified into several categories and analyzed. The paper has shown the areas of research within Cloud based testing that have been done by answering the questions that were defined initially. Eventual bugs and defects reduce application functionality, do not look vocational, and disturb company’s reputation. Thence, radically testing is very important to conduct. At that way, the defects can be discovered and repaired. Even if customers are dissatisfied with a product, they will never recommend that product, so product’s cost and its popularity at the market will decrease. Besides, customer testing is also very important to conduct. Through this process one can find out if application’s functions and characteristics correspond to customers, and what should be changed in application to accommodate it according to customer’s requests. The overall result shows that this methodology is a promising approach for fully automatic test case generation for the testing technique that use condition coverage of the source code. To increase the efficiency and effectiveness, and thus to reduce the overall development cost for software based systems a systematic and automatic test case generator is required. Improved efficiency notwithstanding, even model-generated test cases require peer review, and here is where we saw our second pleasant surprise: with no changes to the original executable software test specification infrastructure, we were able to write an XSLT transform that could convert the runtime XML data generated by the model into the same HTML that previously rendered IML data. The result was that even model-generated test cases could be reviewed the way data-driven test specifications had been; so all the benefits of peer review accrued to model-based as well as data-driven testers. Indeed, given the additional data provided by the model (and only occasionally provided by testers who manually specified test cases), the review of model-based test cases was better than reviews of most data-driven test specs. Many technical difficulties must be overcome before unit testing can become pervasive within climate models. Chief among these is the extreme difficulty of introducing fine-grained tests into the procedures typical of legacy science code. In many cases, this issue can by sidestepped by implementing changes and extensions to models as new modules. This is in contrast with the common practice of “wedging” changes directly into old procedures and ultimately compounding the legacy burden. Fixes to software bugs are a particularly important case to follow this approach, as the constructed test serves as protection against re-introduction of the defect. In the long term more powerful tools must be developed which allow developers to efficiently extract disjoint bits of functionality from the legacy layer.

10. REFERENCES


Author Profiles

Dr.Vidhushi Sharma born in Aligarh, UP. Dr.Vidhushi Sharma Working as a Professor (CSE), in Aligarh Muslim University, Aligarh, Uttar Pradesh, India. He Guided Number of Ph.D Scholars in Aligarh Muslim University & Published number of Journals.
Email: dr.vidhushisharma@rediffmail.com

N.Sudheer born in Ongole Prakasam Dt, AP. Presently he is Pursuing Ph.D (CSE), in Aligarh Muslim University, Aligarh, Uttar Pradesh, India. My Research interest includes Software Engineering and Testing Methodologies & Published number of Journals.
Email:nidamanuri.sudheer@gmail.com