Journal of Engineering Design and Analysis
Volume 6, Issue 1 - 2023, Pg. No. 30-35
Peer Reviewed Journal

Review Article

Ordering Program Elements According to Testing Requirements

Animesh Srivastava[1], Parveen Kumar Saini[2]

[1,2] Chandigarh University, Mohali, Punjab, India.

I N F O

Corresponding Author:
Animesh Shrivastava, Chandigarh University, Mohali, Punjab, India.
E-mail Id:
er.animesh10@gmail.com
Orcid Id:
https://orcid.org/0009-0006-2045-9275

How to cite this article:
Srivastava A, Saini PK. Ordering Program Elements According to Testing Requirements. *J Engr Desg Anal* 2023; 6(1): 30-35.

Date of Submission: 2023-03-25
Date of Acceptance: 2023-04-03

A B S T R A C T

Finding flaws in a software product is the goal of the testing process. However, even after successfully completing the testing step for the majority of practical systems, it is impossible to ensure that the program is error-free. This is a result of the vast input data domain found in the majority of software applications. It is not realistic to test the software in every possible configuration that the input data might take. Even with this real-world constraint on the testing process, its significance shouldn't be understated. It must be kept in mind that testing does reveal numerous flaws in a software program. Testing thus offers a useful method of lowering system flaws and boosting users' confidence in a built system. A few flaws typically persist even after a program has undergone extensive testing. Usually, these remaining flaws are dispersed across the code. It has been noted that flaws in some areas of a program can lead to failures that are both more frequent and more severe than those in other areas. The statements, methods, classes of an object-oriented program should thus be able to be arranged according to how likely they are to result in errors. After the program's components are arranged, the testing effort can be distributed so that the components that frequently fail are tested more. In this method, a program's intermediate graph representation is exploited. A forward slice of the graph is used to estimate a class's influence. Applications for our suggested program metric include coding, debugging, test case design, maintenance, among others.

Keywords: Testing, Software Testing, Test Case, Prioritization, Object Oriented Programming

Introduction

A variety of techniques for prioritising and selecting test cases and influencing nodes have been proposed and empirically investigated in Survey.[1,2] The current knowledge in these areas. According to the findings, most existing methodologies apply structural or functional coverage requirements in relation to the source code run during test cases. One way that our ideas differ from those in the literature is in this area.

Even though testing software is an expensive task in and of itself, the cost of releasing software without testing could be much higher, especially if it affects people's safety. Software testing is any action that looks at a programme or system's feature or ability to see if it does what it's supposed to do and if it does it well. Even though software principles are important to the quality of software and are often used by both programmers and testers, software testing is still thought of as an art. Software testing is hard because of how complicated software is. We can't test a programme well if it's not very complicated. Debugging is just one part of the testing process. Testing can be done to make sure something is reliable, to make sure the quality is good, or to verify and validate. Testing can also be used as a general way to measure something. Testing for correctness and testing for reliability are two of the most important types of testing. When testing software, you have to choose between cost, time, quality.

The earlier problems are found and fixed over the software life cycle; the less money is spent doing so. Our daily lives are becoming more and more infused with software solutions. Software firms are under tremendous pressure to deliver extremely reliable products with very little tolerance for errors. In order to find all flaws, software products are typically tested on several levels. the computer programme. Even after successfully completing the testing process, it is impossible to ensure that a software product is error-free for the majority of practical systems. This problem is caused by the fact that the input data domain of most software products is very large.

Additionally, both time and budget constraints apply to every software product development effort. it is not possible to fully test a piece of software by giving it every possible value for the input data. At the moment, testing takes up an average of half of all development costs and time.[10]

So, it's unlikely that the amount of testing will be done even more. Traditional testing methods are used to test each part of the software product thoroughly. This means that bugs in the software are spread out evenly. But when bugs are present in some parts, they cause problems that are worse and happen more often than in other parts. For instance, if a statement makes important data that many other statements need, then a mistake in this statement would affect many other statements. So, our goal is to figure out which parts of a programme are the most important and need to be tested more thoroughly. We say that an element's influence is the measure of how important and serious it is. We came up with a way to measure how important a statement is and how important a method is. With these two measurements, we can figure out how important a class is. Characterizing code can help with designing, writing, testing, maintaining software. We use the Extended System Dependent Graph to show how code works in the middle. So, it doesn't look like the work of testing could be done any better. Since testing is a sample, it is always important to choose what to test and what not to test, as well as how much to do. The majority of systematic test methods, such as white box testing or black box methods such as equivalence partitioning, boundary value analysis, or cause-effect graphing, generate an excessive number of test cases.[7]

Motivation for our Work

As computers and software are frequently utilised in crucial applications, a flaw might have disastrous results. Huge losses may result from bugs. Critical system bugs have led to plane catastrophes, allowed the space shuttle's systems to fail, suspended stock market trade. A bug can kill. Disasters can be caused by bugs.

Software's dependability and quality are crucial in a society where everything is computerised. Only by conducting rigorous testing can this be possible.

In today's generation lives on the internet and uses IoT devices, we are using firmware, which is important for medical to agriculture. Everywhere we are using software for analysis and for decision making, so reliable software is needed every hour.[9,11,12,13,14]

Objective of our Work

The impact of various programmed elements on the overall reliability of a programmed varies significantly. The influence of different components must therefore be described, the more trustworthy components must undergo extensive testing. to identify undiscovered errors based on specifications. Make sure the product is clear of bugs before shipping or releasing. “Quality is Guaranteed.”

The primary goal of our research is to create effective algorithms to determine how a statement, a method, a class affect an object-oriented programmed. The goal of our work is to find and fix software bugs as early as possible in the software development process, so that software doesn't break down often or badly.

The goal of our work is to identify and isolate software faults at the earliest possible phases of the software development cycle in order to prevent frequent and serious software failures. Our objective is to reduce the failure rate of a system while staying within the testing budget. Two elements are included in the test plan for this.

1. The most crucial components of the application ought to be tested first.
2. Thoroughly test the sections of the code where the presence of a single defect increases the likelihood of failure.

The first one can be identified by taking a look at function visibility, usage frequency, potential failure costs. For the second one, we've come up with an algorithm to find the important parts of the source code.[2]

Related Work

A lot of papers have been written about how to order test cases.[4,5] But not much has been said about the work done to figure out which parts of the code are the most important. Before test cases are made, not much research has been done on how to improve testing. In this field, one area of study is how to make software that can be tested. This work tries to explain how to make software that is easy to test and, hopefully, cheaper to test.

During the design phase of this work's development life cycle, the preconditions, postconditions, assertions for each module are chosen. The other part of pre-testing is setting priorities for testing code. I suggested a way to figure out priorities that puts the most important parts of the code that need to be tested at the top of the list and makes them stand out. This would be a quick way to improve code coverage. Code coverage is a metric that shows how much of the source code of an application is run when its unit tests are run. In theory, the better the code works, the more of it is covered. But 100% code coverage doesn't mean that an app is bug-free in real life. To figure out how powerful an object is, you look at how many other objects in the given programmed use it directly or indirectly. A method on an object can sometimes tell other objects in a programmed what to do by the value it sends back. So, an object's power depends on how many other objects in the programmed depend on it for both control and data, either directly or indirectly. First, we use the source code to make an intermediate representation called a "control dependence graph." Then, we run the programmed using the given set of data. We show our proposed algorithm, which can figure out the influence value of any object and get the dynamic slice of any object at any execution point. Prioritized testing is part of what we do to make sure that the testing process makes great software within the testing budget. In this section, we focus on research results that were reported in the context of prioritization techniques, at the time of test case selection in test suites, or before test cases were built.[2]

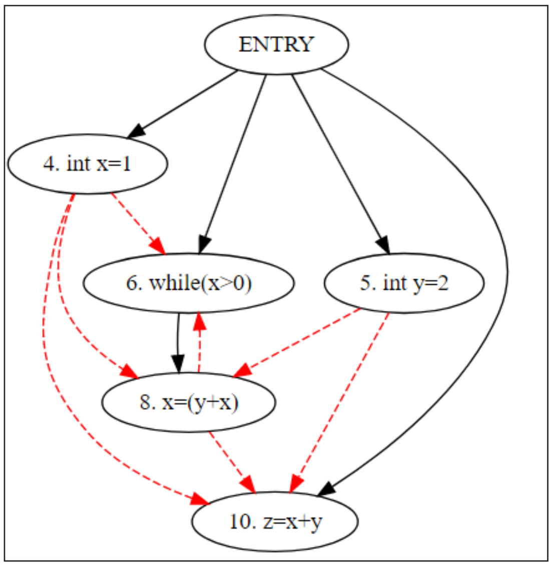

Figure 1.Program Control Dependence Graph

Control Flow

The control flow graph (CFG) is a programmed representation that can be used as a step in several optimization code transformations, including common subexpression removal, copy propagation, loop invariant code movements.

Via the Program Control Dependence Graph, we find the dependence and which affects others. This is one of the ways we can use to find the dependence and affect each other.

Profiling

When we profile a programmed, we can observe where it spent its time and which functions it called when they were active. This data can highlight the areas of the programmed that are running more slowly than anticipated and may benefit from rewriting to speed up execution. Additionally, it can reveal which functionalities are being used more or less frequently than you anticipated. You might be able to see bugs that you wouldn't have otherwise.

Proposed Methods

Earlier work might take a long time (a month or a year), depending on how big the test suite is and how long it takes to run each test case.

But if testers use an effective prioritization technique, they can change the order of the test cases to find faults more often. The method described in this paper used a prioritization algorithm to set the order of the test cases. The goal was to find as many bugs as possible during the constrained execution.

Ordering of Features in a Program

In this section, we explain how we decide the order of programmed elements based on how thoroughly they should be tested. First, we explain how we plan to do things. After that, we show how we calculate the effect of the statement, the effect of the method, the effect of the class.

A Review of Our Methodology

An object-oriented program's classes are made up of code. Every class, it is assumed, contains variables and methods. A class's influence is made up of the cumulative effects of all of its component parts. As a result, we assess the impact of each statement and, if a statement calls a method, we assess the impact of the method as well. Our approach ignores variable values and is based on a static analysis of the code. As a result, it has trouble handling loops and recursive function calls. Class has the same impact as the sum of the impacts of all applicable assertions and procedures. This technique determines a class's effect statically. We first go over how to determine the influence of a remark, then influence of a method and influence of a class are discussed.

Influence of a Statement

The output of one statement in a programme could be dependent on the output of another one. The statement is more critical if the influence is higher. The number of other statements in the supplied programme that directly or indirectly use that variable determines the statement's influence. Given that there is no call vertex, we provide a metric to calculate influence. If a statement is designated as a vertex, its influence will be determined independently using the method metric's influence, then added to determine the desired statement's overall influence. The following factors are used to determine the statement's percentage influence:

$$\frac{\text{Total number of impacted nodes}}{\text{The total number of nodes in the graph}} \times 100$$

Algorithm

Input: The code for the programme and the statement.

Outcome: Effect of the given statement.

StmtInfluence(statement) 1. Build the program's ESDG in a static way.

2. The for statement should go through all of the edges that depend on it and mark them.

3. Repeat step 2 for each of the marked nodes until there are no more edges that depend on them.

4. If a marked node is a call vertex, use MethodInfluence to figure out how important it is (callvertex).

5. Count the marked nodes and use an expression to figure out the influence (1).

6. Stop.

}

Influence of a Method

When a method in a programme works out a result, that result affects the other methods and statements. One method can have an effect on another method or statement in the programme. If the method has a bigger effect, then it is more important. We have made a metric for object-oriented programmes called "the influence of a method."

The impact of a method is measured by how many other statements and methods in a given programme use the method's results directly or indirectly.

If the method we want to find the influence of calls other methods, the total influence of the method will be the sum of the influence of the method itself and the influence of the methods it calls.

The percentage of a method's effect can be found by:

$$\frac{\text{Total number of impacted nodes}}{\text{The total number of nodes in the graph}} \times 100$$

Algorithm

Input: The name of the method of a programmed and the name of the method of the programmed.

Results: What the method did.

MethodInfluence (callvertex){

1. ESDG should be a part of the programmed.

2. Go through all the edges and mark the ones you've already been to for the method's starting point.

3. Go through all of the edges of each node you visit and label the node it belongs to as visited, if it isn't a call-vertex node, mark it as influenced if you haven't already.

4. Determine whether each visited node is a call vertex. If so, proceed along the call's edge and do the following:

(a) Call the vertex and walk through each polymorphic edge if the following node is polymorphic and add the matching node to a queue Q

b) If it doesn't, add the node to Q.

5. Remove the nodes from Q. Mark the affected node, then repeat steps 2–4 for that node.

6. Step 5 should be repeated until there is nothing in Q.

7. Go through each node that has been marked as influenced and mark each of its edges as influenced.

if it hasn't been done already.

8. Use the phrase to figure out what the method will do (2).

9. Stop.

}

Influence of a class

The influence of a class is the sum of the effects of all the other parts of a given programmed that use the results of the class in some way. We count how many nodes are affected. The MethodInfluence(callvertex) metric is used to figure out the influence of nodes that contain function calls. The StmtInfluence (statement) metric is used to figure out the influence of all other statements.

The influence of a class is given as:

Total number of impacted nodes

$$\overline{\phantom{\text{Total number of impacted nodes}}} \times 100$$

The total number of nodes in the graph

Algorithm

Input: A sample programmed and the class's name.

Outcome: How the class changed things.

```
ClassInfluence(classname)
{
1. Build the program's ESDG in a static way.
2. Move through the class entry vertex to each member of the class and mark each one as visited.
3. For each visited node, go through all of its edges and mark the corresponding node as visited. If the node is not a call-vertex, mark it as influenced if it hasn't already been marked.
4. Check each visited node to see if it is a call vertex. If it is, use MethodInfluence to figure out how important this statement is (callvertex).
5. For each node that has been marked as being influenced, go through all of its edges and mark each one as being influenced if it hasn't already been done.
6. Use the expression to figure out the influence of the given class (3).
7. Stop.
}
```

For experimental studies' purposes, we have taken a sample programmed that has 134 nodes.

Table 1.Experimental Studies

The sum of all nodes that were impacted	The Program's Total Nodes	Relevance percentage
96	134	71.64
22	134	16.41
8	134	5.9
7	134	5.2
1	134	.7

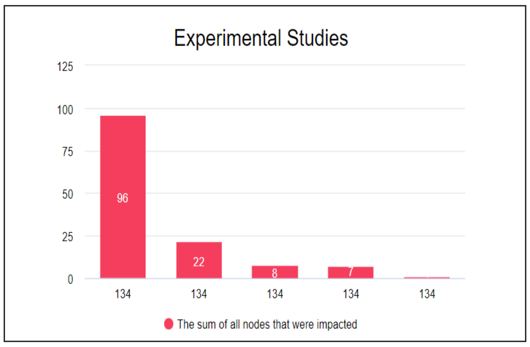

Figure 2.Experimental Studies

We have found the most influenced node, which impacts the highest node, so we have to take that node first in the testing. For this, we must identify the most influenced statement, method, class and test accordingly.

- Prioritizing testing can help with a variety of aims, as illustrated below
- Obtaining high-risk errors discovered early in the testing process
- To increase the likelihood that specific code modifications may create mistakes early in the testing process
- To increase the frequency with which code that can be covered gets covered
- To increase the reliability of a system

Conclusion

We made a programme metric that looks at how important programme elements are. The influence shows which parts of the programme are affected more than others. So, the factors with more influence are more important, including them will make it more likely that the software will fail. So, the intended metrics help a lot in figuring out which parts are the most important and tell us to be very careful when building the parts that have the most impact during the software development cycle. This shows that testing the parts that aren't as important can be done with fewer test cases than testing the parts that are more important. This saves time for testing the parts that are more important. It is based on a program's static analysis.

- This is helpful when creating and ranking test cases
- Understanding the impact of individual programme elements is helpful. Due to this, we have more trustworthy components with which to perform rigorous testing

References

1. Prioritization of Program Elements Based on Their Testing Requirements, Computer Science and Engineering, Kanhaiya Lal Kumawat, National Institute of Technology Rourkela (2009)
2. Reliability Improvement Based on Prioritization of Source Code, Mitrabinda Ray and Durga Prasad Mohapatra, Department of Computer Science and Engineering, National Institute of Technology Rourkela (2010)
3. Danjun Zhu, Gangtian Liu, "Deep Neural Network Model-Assisted Reconstruction and Optimization of Chinese Characters in Product Packaging Graphic Patterns and Visual Styling Design", Scientific Programming, vol. 2022, Article ID 1219802, 12 pages, 2022. https://doi.org/10.1155/2022/1219802
4. S. M. Guertin. Board Level Proton Testing Book of Knowledge for NASA Electronic Parts and Packaging

Program. Accessed: Oct. 2018.
5. S. M. Guertin, “Lessons and recommendations for board-level testing with proton,” in Proc. Small Satell. Conf., Logan, UT, USA, 2018.
6. A. Coronetti et al., “Radiation Hardness Assurance Through System-Level Testing: Risk Acceptance, Facility Requirements, Test Methodology, Data Exploitation,” in IEEE Transactions on Nuclear Science, vol. 68, no. 5, pp. 958-969, May 2021, doi: 10.1109/TNS.2021.3061197.
7. Jordi Roglans-Ribas, Kemal Pasamehmetoglu & Thomas J. O’Connor (2022): The Versatile Test Reactor Project: Mission, Requirements, Description, Nuclear Science and Engineering, DOI: 10.1080/00295639.2022.2035183
8. E. W. Dijkstra. 2022. On the Reliability of Programs. Edsger Wybe Dijkstra: His Life, Work, Legacy (1st ed.). Association for Computing Machinery, New York, NY, USA, 359–370. https://doi.org/10.1145/3544585.3544608
9. Srivastava, A., Kumar, A. (2022). A Review of Network Optimization on the Internet of Things. In: Saini, H.S., Sayal, R., Govardhan, A., Buyya, R. (eds) Innovations in Computer Science and Engineering. Lecture Notes in Networks and Systems, vol 385. Springer, Singapore. https://doi.org/10.1007/978-981-16-8987-1_6
10. N. Srivastava, U. Kumar and P. Singh (2021) Software and Performance Testing Tools. Journal of Informatics Electrical and Electronics Engineering, Vol. 02, Iss. 01, S. No. 001, pp. 1-12, 2021. https://doi.org/10.54060/JIEEE/002.01.001
11. Kumar, G., Singh, G., Bhatanagar, V., & Jyoti, K. (2019). SCARY DARK SIDE OF ARTIFICIAL INTELLIGENCE: A PERILOUS CONTRIVANCE TO MANKIND. Humanities & Social Sciences Reviews, 7(5), 1097-1103. https://doi.org/10.18510/hssr.2019.75146
12. Gupta, R., Bhatnagar, V., Kumar, G., & Singh, G. (2022). Selection of suitable IoT-based End-devices, tools, technologies for implementing Smart Farming: Issues and Challenges. International Journal of Students’ Research in Technology & Management, 10(2), 28-35. https://doi.org/10.18510/ijsrtm.2022.1024
13. Singh, G. and Yogi, K.K. (2022a). Internet of Things-Based Devices/Robots in Agriculture 4.0. In: Karrupusamy P., Balas V.E., Shi Y. (eds) Sustainable Communication Networks andApplication. Lecture Notes on Data Engineering and Communications Technologies, vol 93. Springer, Singapore. https://doi.org/10.1007/978-981-16-6605-6_6
14. Singh, G. and Yogi, K.K. (2022b). Usage of Internet of Things Based Devices in Smart Agriculture for Monitoring the field and Pest Control. 2022 IEEE Delhi Section Conference (DELCON), pp.1-8. https://doi.org/10.1109/DELCON54057.2022.9753021