

# **TOOL SUPPORT FOR AUTOMATION OF C++ TEST CASE GENERATION**

Imaran Shyabith Maher Dickwella

(168217P)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

February 2020

# **TOOL SUPPORT FOR AUTOMATION OF C++ TEST CASE GENERATION**

Imaran Shyabith Maher Dickwella

(168217P)

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree Master  
of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

February 2020

## DECLARATION

I declare that the content of this research is my own work and the CS5999 PG Diploma Project Report does not include any content previously submitted for a Degree or Diploma in any other University or institute of higher learning without acknowledgement and to the best of my knowledge and belief, this report does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to regenerate and distribute my thesis/dissertation, in whole or in part in print, electronic or other media. I retain the right to use this content in whole or part in future works (such as articles or books).

.....

.....

Imaran Shyabith Maher Dickwella

Date

The above candidate has carried out this CS5999 PG Diploma Project Report under my supervision.

.....

.....

Dr. Indika Perera

Date

## **ABSTRACT**

Testing of software plays a vital part in the software development process. It is the phase in the software development life cycle that make sure the developed software is functionally correct. Software faults can be expensive when the fault is found at the production system therefore it is essential to find software errors at the development stages of the project when the cost is minimum. Testing makes sure the software which is developed covers functional and nonfunctional requirements.

Unit tests take an essential place in software testing and it is the earliest test a developer or a tester can perform on the implementation, defects can be detected at early stages, and fixes can be done with lesser cost due to lesser dependencies. Even though it is essential to implement unit tests, it is not the case in reality. Most software companies rely heavily on end to end testing. The main reason for lack of testing at the unit test level is due to its time-consuming nature and it will not provide functional coverage whereas by performing end to end tests, end-user requirements can be captured and tested. In reality, approximately two-thirds of the development time is spent on unit testing related activities and the rest of the time is spent on designing and implementation. However, due to time constraints and budget limitations, allocating a large portion of the time for unit testing is not practical.

In this research, I came up with a solution to address the above-mentioned issues and to eliminate the requirement of writing unit tests manually. I implemented a tool which generates unit tests for applications that are implemented in C++. The process is completely automated and the unit test files generated by the tool are human-readable. Developers no longer need to implement unit tests however they may have to validate the correctness of the generated unit tests and as desired they can extend the meaningfulness of the generated unit tests.

The tool is embedded with three test data generation mechanisms; Random Value Generation, Goal Oriented Test Generation and Feedback driven Test Data Generation. The tool successfully produced test data to attain approximately 95% of code coverage with data generation mechanisms except Random value generation in multiple test experiments however when the test unit has a higher number of branches Feedback driven test data generation mechanism showed better performance as it took lesser time for data generation. Time growth is exponential when the number of branches gets increased in Goal Orientated Test Generation. This tool can be extended for complex structures and I can successfully conclude that the research is successful as the tool showed higher accuracy.

## **ACKNOWLEDGEMENT**

My sincere thanks go to my project supervisor Dr. Indika Perera for guiding me throughout the project and giving a clear vision and a mission to accomplish during this period. It is essential to mention my friends and the staff of MillenniumIT software engineering Pvt Ltd for supporting me throughout the MSc course.

I take this moment to thank my parents for the support and the encouragement given to follow the MSc. in Computer Science amidst this busy schedule at working place. Last but not least, my sincere thanks go to the Head of Department of Computer Science and Engineering and the members of the academic staff for advising and guiding me during project evaluations and presentations.

## Table of Contents

DECLARATION	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
1 INTRODUCTION	2
1.1 Background	2
1.2 Unit Testing and Test Case Generation	5
1.2.1 Definition of unit testing	5
1.2.2 Disadvantages of manual unit tests generation	5
1.2.3 When to use unit test generation	6
1.2.4 Unit testing types	7
1.2.5 Other uses of unit testing	10
1.3 Motivation	11
1.4 Research Problem and Objectives	12
2 LITERATURE REVIEW	14
2.1 Program versus Specification-based Test Generation	14
2.2 Code Based Test Generation Methods	14
2.2.1 Static structural test data generation	15
2.2.1.1 Symbolic execution based technique	15
2.2.1.2 Domain reduction	16
2.2.1.3 Search-based techniques	16
2.2.2 Dynamic structural test data generation	16
2.2.2.1 Random selection	17
2.2.2.2 Applying local search	17
2.2.2.3 Goal oriented test generation	18
2.3 Comparison of Test Generation Approaches	19
2.4 Tools	19
2.4.1 KLOVER	19
2.4.2 KLEE	20
2.4.3 CREST	21

2.4.4	CAUT	21
2.4.5	DART	22
2.4.6	CUTE	22
2.4.7	PathCrawler	23
2.4.8	AgitarOne	24
2.4.9	CATG	24
2.4.10	EvoSuite	24
3	METHODOLOGY	27
3.1	Proposed Solution	27
3.1.1	High-level design	27
3.2	Scrutiny of the Solution	29
3.3	Progress	31
3.4	Evaluation Methodology	31
4	SYSTEM ARCHITECTURE AND THE IMPLEMENTATION	33
4.1	System Overview	33
4.1.1	Infrastructure layer	33
4.1.2	Application layer	33
4.1.3	Functional layer	33
4.2	Random Value Generator	34
4.3	Feedback Driven Value Generator	35
4.4	Goal Oriented Value Generator	36
4.5	Architecture of Code Generator	38
4.5.1	Code parser	39
4.5.2	Fitness function evaluator module	41
4.5.3	Analyzer	43
4.5.4	Code generator	44
4.5.5	Report generator	45
4.6	Parameters	45
5	EVALUATION	47
5.1	Overview	47

5.2	Evaluation of the Tool	47
5.2.1	Evaluation of feedback driven unit test generator	48
5.2.1	Evaluation of goal oriented test generator	52
5.3	Algorithm Comparison	55
5.3.1	Comparison of code coverage	61
5.3.2	Comparison of time	62
5.3.3	Proof of concept	63
5.3.4	Line coverage	66
5.3.5	Branch coverage	66
6	CONCLUSION	69
6.1	Summary	69
6.2	Contribution	71
6.3	Limitations	72
6.4	Future work	73
	Reference	74
	Appendix A – Random Test Generation Method	77



## List of Figures

Figure 1 - Test pyramid .....	3
Figure 2 - Sample library interface.....	6
Figure 3 - Sample function implementation.....	6
Figure 4 - Sample Algorithmic function .....	7
Figure 5 - Sample Test Case.....	7
Figure 6 - Sample class implementation .....	8
Figure 7 - Expected test class .....	9
Figure 8 - Sample function.....	9
Figure 9 - Test generation flow .....	11
Figure 10 - Overall architecture of KLOVER [12] .....	20
Figure 11 - High level design .....	28
Figure 12 - Function to be tested in KLEE engine.....	29
Figure 13 - Marking symbolic variables for KLEE engine.....	30
Figure 14 - Unit test generator implementation .....	33
Figure 15 - Sample function II .....	35
Figure 16 - Execution tree .....	36
Figure 17 - Sample function III.....	36
Figure 18 - Unit test Module chain.....	38
Figure 19 - Field class .....	39
Figure 20 - Parameter class .....	40
Figure 21 - Method class .....	40
Figure 22 - Constructor class.....	41
Figure 23-Branch Distance computation .....	42
Figure 24-Sample function for fitness calculation.....	42
Figure 25 - Derived Fitness Function for Figure 27.....	42
Figure 26 - Generated unit test block .....	43
Figure 27 - Generated Test file.....	44
Figure 28 - Header Class .....	48
Figure 29 - Implementation class with std data structures .....	49
Figure 30 - LCOV report for the data structure class.....	50
Figure 31 - Time analysis for the data generation.....	51
Figure 32-Class I.....	52
Figure 33 - Header file for the above class.....	52
Figure 34 - Coverage report for class in figure 30 .....	53
Figure 35 - Time measures for Class I with Goal Oriented Test data.....	54
Figure 36 - Unachievable branch statements.....	54
Figure 37 - Class A.....	55
Figure 38 - Random Generator Result for Class A .....	55

Figure 39 - Goal Oriented Result for Class A .....	55
Figure 40 - Class B .....	56
Figure 41 - Goal Oriented Result for Class B .....	56
Figure 42 - Random Generator Result for Class B.....	56
Figure 43 - Class C .....	57
Figure 44 - Goal Oriented Result for Class C .....	57
Figure 45 - Random Generator Result for Class C.....	57
Figure 46 - Goal Oriented Result for Class D.....	58
Figure 47 - Random Generator Result for Class D .....	58
Figure 48 - Class D.....	58
Figure 49 - Class E .....	59
Figure 50 - Random Generator Result for Class E.....	59
Figure 51 - Goal Oriented Result for Class E.....	59
Figure 52 - Random Generator Result for Class F.....	60
Figure 53 - Class F .....	60
Figure 54 - Goal Oriented Result for Class F.....	60
Figure 55 - Code Coverage Comparison Table.....	61
Figure 56 - Time Analysis Table.....	62
Figure 57 - Pseudo code for search logic .....	63
Figure 58 - Example for proof of concept case 1 .....	64
Figure 59 - Example for proof of concept case II .....	65
Figure 60 - Proof of concept case III.....	65
Figure 61 - Time Comparison Vs Branches .....	67

## List of Abbreviations

Abbreviation	Description
CUT	Class under test
E2E	End to End
API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
SO	Service Orientation
SOA	Software Oriented Architecture
TDD	Test Driven Development
GUI	Graphical User Interface
GTest	Google Test
GMock	Google Mock
CFG	Control Flow Graph
SUT	System under test