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

List of Figures

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