Design and Implementation of Combinatorial Testing based Test suites for Operating Systems used for Internet of Things

THESIS

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

ABHINANDAN H PATIL

Under the Supervision of

Prof. NEENA GOVEAS

and Co-supervision of

Prof. KRISHNAN RANGARAJAN

IJSER

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

2019

BIRLA INSTITUTE OF TECHNOLOGY AND

SCIENCE, PILANI

Certificate

This is to certify that the thesis entitled **'Design and Implementation of Combinatorial Testing based Test Suites for Operating Systems used for Internet of Things'** and submitted by **ABHINANDAN H PATIL**, ID.No. **2013PHXF0408G** for award of Ph.D. of the Institute embodies original work done by him under our supervision.

Signature of the Supervisor :
Name in capital letters : **Prof. NEENA GOVEAS**
Designation : **PROFESSOR**
Date :

IJSER

Signature of the Co-supervisor :
Name in capital letters : **Prof. KRISHNAN RANGARAJAN**
Designation : **PROFESSOR**
Date :

Declaration

I, Abhinandan H Patil, declare that this thesis titled, 'Design and Implementation of Combinatorial Testing based Test Suites for Operating Systems used for Internet of Things' submitted by me under the supervision of Prof. Neena Goveas and Prof. Krishnan Rangarajan is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for award of any degree.

IJSER

Signature of the student:

Name of the student: **ABHINANDAN H PATIL**

ID number of the student: **2013PHXF0408G**

Date:

Abstract

Regression test suites are maintained by software system developers to ensure that any new code development does not affect existing functionalities. Such test suites need to be optimal in size so as to balance the requirements of maximum coverage and minimum execution time. For software with multiple input parameters and configurations, Combinatorial Testing (CT) is a method which can be used to generate regression test suites.

Regression test selection, augmentation, prioritizing and pruning are the areas in which significant amount of research work has already been done. In spite of this, for most software systems, where multi parameters are involved either in the configuration or in the input parameters of the regression test suite the combinatorial testing is not explored. We explore the combinatorial testing methodology and apply it to regression test suite of case study operating system Contiki in this thesis.

Thesis starts with the test suite execution timing analysis.

Thesis then proposes an integrated test environment approach which brings about a better integration of different tools which are available.

The thesis continues with a demonstration of generating effective test suite for multiparameter software using Advanced Combinatorial Testing for Software (ACTS) Tool and its verification using Code Coverage Tools. The Thesis details generation of the regression test suite using a methodology called as CT based Regression Test Suite (CT-RTS). This is applied to a free software called College Time Table (CTT).

To demonstrate the advantages of use of CT-RTS the Thesis then does a detailed study of Contiki, an IoT Operating System. The Internet of Things technology deals with connecting devices, called as things, to Internet via standardized networking protocols. The networking protocol is part of the Operating System (OS) deployed on the nodes or motes of Internet of Things. In addition to networking capabilities, the OS needs to meet the requirements of extended battery life, memory constraints etc.. Since it is an evolving piece of software, the testing of the OS has to be thorough and streamlined. Details of the existing regression test suite of Contiki is presented. The Thesis lists

IJSER © 2019
http://www.ijser.org

some of the limitations of the existing regression test suite by studying the code coverage.

To overcome these limitations, the Thesis proposes two methods for generation of test suite using CT-RTS: One is to augment the existing test suite. Second mechanism is to create a new test suite.

The Thesis shows the effectiveness of the CT-RTS by applying it to Contiki Operating System and its Cooja simulator. The utility of the CT-RTS has been demonstrated by solving a research problem in the area of Regression test suite creation. The CT-RTS used to create a functional Regression test suite creation mechanism which can be used for large multiparameter software.

The Thesis shows that it is possible to take a more rigorous approach to the problem of Regression test suite creation using the CT-RTS approach.

The Thesis makes the following research contributions.

1. Study of generic regression test suite is done in this phase. Regression test suite execution timing analysis methodologies are studied using the statistical approach.

2. An integrated test environment approach for combinatorial testing is proposed. It is a centralized approach which reduces the number of tools and duplicated functionality leading to better integration of different tools making maintenance of the test setup simple.

3. Thesis demonstrates generation of a regression test suite for a multi parameter software using a proposed methodology called CT-RTS. The effectiveness of the test suite is verified using the traditional code coverage metrics.

4. Thesis proposes the test design methodology for Internet of Things operating systems. Extraction of parameters by studying the existing regression test suite, the execution and use of the software is demonstrated.

5. CT-RTS approach is applied to generate regression test suites for the Contiki OS and Cooja simulator. Comparison of the effectiveness of the existing test suites, re-architectured test suites and new test suite designed using CT-RTS is done.

6. The Thesis proposes a mechanism for automation of test scripts generation. This has been successfully used to generate functional test suites for Contiki OS and Cooja simulator.

IJSER

7. Residual test coverage algorithm is enhanced for prioritization of the regression test suite using the code coverage and execution time as the input parameter. Further, black box approach to test suite prioritization using statistical techniques is proposed.

IJSER

IJSER © 2019
http://www.ijser.org

Acknowledgements

Grateful to God, My Family members, Well wishers and My Teachers in that order.

Abhinandan H. Patil

.

IJSER

IJSER © 2019
http://www.ijser.org

Contents

Certificate......2

Declaration......3

Abstract......iv

Acknowledgements......vii

Contents......viii

List of Figures......xv

List of tables......xvi

Abbreviations......xvii

1 Introduction......19

1.1 Background......19

1.1.1 IoT Operating Systems......19

1.1.2 Testing for IoT Operating Systems......19

1.1.3 Regression testing......20

1.2 Motivation......20

1.3 Problem statement......21

Objective 1: Study of Regression test suites in general......21

Objective 2: Design and implementation of combinatorial testing based test suites for internet of things operating system and its simulators......21

Objective 3: Measuring the effectiveness of designed test suites using the traditional coverage techniques like code coverage......22

Objective 4: Automation of test scripts generation from combinatorial testing design model and analyzing coverage to refine the combinatorial testing design model......22

Objective 5: Propose an integrated test environment for combinatorial testing......22

1.4 Research goals......22

IJSER

IJSER © 2019
http://www.ijser.org

Research goal 1: 22
Research goal 2: 23
Research goal 3: 23
Research goal 4: 23
Research goal 5: 23
Research goal 6: 23
1.5 Solution Approach 23
1.6 Publications 24
1.7 Research Contributions 25
1.8 Thesis outline 26
Chapter 2 – Literature Survey 26
Chapter 3 - Regression Test Suite Execution Time Analysis using Statistical Techniques 26
Chapter 4 - Integrated Test Environment for Combinatorial Testing 26
Chapter 5 - CT-RTS: Advanced Combinatorial Testing for Software Regression Testing 27
Chapter 6 –CT-RTS: Generating Effective Test Suite for Multiparameter Software using ACTS Tool and its Verification using Code Coverage Tools 27
Chapter 7 - Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach 27
Chapter 8 - Test Suite Design Methodology using Combinatorial Approach for Internet of Things Operating Systems 27
Chapter 9 – CT-RTS: Contiki and Cooja Regression Test Suites Design and Implementation using Combinatorial Testing 28
Chapter 10- CT-RTS: Combinatorial Testing based Regression Test Suite: Functional Test Case Generator for Contiki and Cooja 28
Chapter 11 - Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques 28
Chapter 12 – Conclusion 28

IJSER

IJSER © 2019
http://www.ijser.org

2 Literature Survey 29
2.1 Properties and Characteristics of IoT Operating Systems...............29
2.2 Importance of Regression Testing..31
2.3 Combinatorial techniques based approaches to Software Testing .31
2.4 The need of Combinatorial based testing techniques for IoT OS 31
2.5 Use of Combinatorial technique based tools 32
2.6 Software Test Coverage and its relevance to the design of Regression test suites 33
2.6.1 CodeCover 33
2.6.2 OpenClover..33
2.7 Gaps in Existing Research 34
3 Regression Test Suite Execution Time Analysis using Statistical Techniques 36
3.1 Introduction...36
3.2 Functional Simulator Tools...37
3.3 Java Functional Simulator Tools..38
3.4 Java Hotspot VM Options 39
3.5 Test Case and Test Execution Time Observations...........................39
3.6 Statistical Techniques for Execution Time Analysis 39
3.7 Limitations of Statistical Techniques..42
3.8 Advantages of Statistical Approach 43
3.9 Conclusion 43
4 Integrated Test Environment for Combinatorial Testing 44
4.1 Introduction...44
4.2 Overview of Integrated Test Environment.......................................45
4.3 Test Model Generator 46
4.4 Test Generator 47
4.5 Test Management Tool 47

IJSER

4.6 Selection and Prioritization Tool 48
4.7 Defect Tracking Tool...49
4.8 Analysis...49
4.9 Model Checking Tool...50
4.10 Conclusion...50
5 CT-RTS: Combinatorial Testing based Software Regression Suite 51
5.1 Introduction...51
5.2 CT-RTS: Readily Executable Test Cases 53
5.3 CT-RTS: Functional Test Case Generation 53
5.4 Conclusion..53
6 CT-RTS: Generating Regression Test Suite for Multiparameter Software and its Verification using Code Coverage Tools..54
6.1 Introduction...54
6.2 Brief Literature Survey of CT 55
6.3 ACTS Tool..56
6.4 Open Clover..56
6.5 College Time Table 57
6.6 CT-RTS: Generating The Test Cases and Gathering Coverage Data .58
6.6.1 ACTS Tool Usage for Generating The Test Cases 59
6.7 Results and Results Analysis..61
6.8 Conclusion..63
7 Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach 64
7.1 Introduction...64
7.2 Contiki Testing Environment 65
7.3 Combinatorial Testing 65
7.4 CodeCover Tool Usage 66
7.5 Results 67

IJSER © 2019
http://www.ijser.org

7.6 Conclusion 67

8 Test Suite Design Methodology using Combinatorial Approach for Internet of Things Operating Systems 68

8.1 Introduction.. 68

8.2 Typical Workflow for Baselining the regression Test Suite............. 68

8.3 Process of Redesigning the Regression Test Suite if it Already Exists 69

8.3.1 Contiki Specific Details 70

8.4 Process of Designing the Regression Test Suite if it Does Not Exist 71

8.5 Contiki Specific Environment Changes to be Done......................... 71

8.6 Conclusion 73

9. CT-RTS: Contiki and Cooja Regression Test Suites Design and Implementation using Combinatorial Testing 74

9.1 Introduction.. 74

9.2 Background.. 74

9.2.1 Existing regression test suite.. 75

9.2.2 ACTS tool for generating combinatorial test design 76

9.2.3 Code coverage using OpenClover 76

9.3 Re-engineering the base test suite 77

9.4 Test design using ACTS tool for re-engineered test suite 79

9.5 Auto generation of test cases 80

9.6 Test design for Cooja test suite using ACTS tool 84

9.7 Code coverage data gathering process 84

9.8 Results 85

9.9 Results analysis.. 91

9.10 Supplementary material.. 93

9.11 Conclusion 93

10. Combinatorial Testing based Functional Test Case Generator for Contiki Operating System and Cooja Simulator 94

IJSER

10.1 Introduction......94
10.2 Combinatorial testing and NIST ACTS tool......96
10.3 Contiki the IoT operating system......96
10.4 Cooja simulator......98
10.5 Regression test suite of Contiki Operating System......98
10.6 Requirements for FTCGCC......99
10.7 High level design of FTCGCC......100
10.8 Software implementation......101
10.8.1 Java's regexp parser......103
10.8.2 Java Document Object Model Parser......103
10.8.3 Data structures and functions......104
10.9 FTCGCC usage in Contiki environment......104
10.10 Conclusion......105
11 Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques......106
11.1 Introduction......106
11.2 Test Coverage Algorithm for White Box Testing......106
11.3 Residual Test Coverage Algorithm enhancements for White Box Testing......108
11.4 Statistical Approach for Prioritization of Test Cases for Black Box Testers......110
11.5 Coverage Tools: CodeCover a case study......111
11.6 Process Flow for Collecting Metrics of Choice......112
11.7 Advantages of Test Suite Prioritization......112
11.7 Conclusion......113
12. Conclusion......114
12.1 Introduction: The research problem......114
12.1.1 Summary of results......114

IJSER

12.2 Conclusions..115

12.2 Future work 116

Appendix A: ACTs Generated Test Design for Contiki Operating System. 117

Appendix B: Code Coverage Data Gathered for Existing Test Suite of Contiki and Cooja using CodeCover 120

Appendix C: Tweaking of Ant build.xml for Gathering The Coverage Data with CodeCover..121

APPENDIX D: ACTS Test Design Input for Re-engineered Test Suite 124

APPENDIX E: ACTS Test Design for Cooja Test Suite 127

Appendix F: Code for Auto Generating csc Files. 130

Appendix G: Candidate's Biography...151

Appendix H: Publications of The Candidate...152

Publications from Thesis 152

Other Publications..153

Appendix I: Supervisors Biodata 154

Appendix J: Co-Supervisors Biodata...155

References..168

IJSER

IJSER © 2019
http://www.ijser.org

List of Figures

Figure 1. Generic Test Setup Involving Simulator Tools in Network.........37
Figure 2. Generic Framework of Simulator Tools Explained. 38
Figure 3. Normal Distribution Curve of Hypothetical Test Suite................40
Figure 4. Normal Distribution Curves for the Same Test Suite on Two Different Setups. 41
Figure 5 Integrated test environment entities. 46
Figure 6. Process flow diagram for CT-RTS 52
Figure 7. Process flow diagram for generating the ACTS test suite for CTT software and measuring the coverage 59
Figure 8. ACTS tool populated data for CTT 60
Figure 10. OpenClover output window at the project level 62
Figure 10. Open Clover output window granular level 62
Figure 11. Typical work flow for base lining the test suite 69
Figure 12. Process of base lining the test suite if it already exists............70
Figure 13 Process of baselining the test suite if it does not exist 73
Figure 14 Process of gathering code coverage for CT 79
Figure 15. Functional test case auto generation tool 80
Figure 16. Test case auto generation process...81
Figure 17. Sample input text file for the tool 82
Figure 18. Sample output XML file 83
Figure 19 Cooja and Open Clover interaction 84
Figure 20. Test environment change for Clover data gathering 85
Figure 21 Reading the treemap...88
Figure 22 Class coverage distribution in simulator for three suites A, B and C respectively. 89
Figure 23. Tree maps of code coverage in simulator for three suites A, B and C respectively 90
Figure 24. Source code analysis of simulator with LOCMetrics 91
Figure 25. Code metrics of Cooja code base 91
Figure 26. Generic structure of csc file 102
Figure 27. Process flow for collecting the metrics of choice 112

IJSER

IJSER © 2019
http://www.ijser.org

List of tables

Table 1. Execution time of the test suite for various JVMs and OSs..........39
Table 2. Tabulation table for co-efficients of correlation..........................42
Table 3. Test generator tools 47
Table 4. Test management tools...48
Table 5 Defect tracking tools...49
Table 6. Parameter and their values in ACTS for CTT software 58
Table 7. Clover coverage data for CTT software 61
Table 8. Input parameters for the ACTs tool...66
Table 9 Test suites and their description 75
Table 10. Code coverage in simulator for test suite A 86
Table 11. Code coverage in simulator packages for Test Suite B 86
Table 12. Code Coverage in simulator package for Test suite C. 87
Table 13. Various existing test case autogeneration tool..........................95
Table 14. Node operating system 96
Table 15. IoT layers and protocols 97
Table 16. Stages and functionality of FTCGCC 100
Table 17. Important data structures and functions.................................104
Table 18. Table for calculating the new metric.......................................110
Table 19. Traceability from research problems to the outcomes 114

IJSER

IJSER © 2019
http://www.ijser.org

Abbreviations

ACTS	Advanced Combinatorial Testing for Software
AST	Applied Statistical Techniques
CCM	Combinatorial Coverage Measurement
CT	Combinatorial Testing
CT-RTS	Combinatorial Testing based Regression Test Suite
IoT	Internet of Things
JVM	Java Virtual Machine
NIST	National Institute of Standards and Technology
NuSMV	New Symbolic Model Verifier
RTCA	Residual Test Coverage Algorithm
RTS	Regression Test Suites
ST	Statistical Techniques

IJSER

IJSER © 2019
http://www.ijser.org

IJSER

IJSER © 2019
http://www.ijser.org

1 Introduction

1.1 Background

The term "Internet of Things" was first proposed by Kevin Ashton in 1999 [1] to describe a system where the Internet is connected to the physical world via ubiquitous sensors. The concept emerged in the context of the developments at the MIT Auto-ID Center on identification technologies. The Internet of Things (IoT) [2] is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment [3].

From the technical point of view, it is about connecting new devices, called objects or things, and investigating the issues related to connecting these objects with the network in order to develop exploitable applications. To tackle these issues, it is important to understand the Operating Systems which are being developed for IoT and to ensure that all the software works as designed on all the objects. Since the software is evolving and dynamic, one of the critical requirements, is to have an effective regression test suite.

IJSER

1.1.1 IoT Operating Systems

IoT Operating Systems consists of Operating systems designed for resource constrained devices which need to be networked. This networking is done using the IP based networking [4]. Several IoT OS's have been developed and many implementations have been successfully demonstrated like Contiki OS which has been used for intruder detection [5] etc.

1.1.2 Testing for IoT Operating Systems

Theoretical foundations for study of software testing has been well established [6]. We start with a study of testing approaches which will work when dealing with IoT OS. There are several challenges when dealing with IoT OSs. One of them is the diversity of the devices in which the OS will be executing. For example, OS like Contiki, RIOT etc. have now been put forward as possible candidates for installation in various processors/devices like MicaZ,

IJSER © 2019
http://www.ijser.org

Exp5438, z1, wismote, sky, esb, etc. [7]. These have very different processors, radios, storage capacity etc. This makes testing of any working property difficult as every combination of these parameters need to tested.

An operating system, additionally, is also an evolving piece of software. Developers design regression test suites to make the process of incorporating changes in the code base trouble free. These regression suites have to play a conflicting role of being comprehensive versus being compact enough to run in the smallest possible time. The duration of these executions are often less than one night to ensure that the development work is not hindered.

1.1.3 Regression testing

The word regress means to return to a previous, usually worse, state. Regression testing refers to that portion of the test cycle in which a program is tested to ensure that not only does the newly added or modified code behaves correctly, but also that code carried over unchanged from the previous version continues to behave correctly. Thus regression testing is useful, and needed, whenever a new version of a program is obtained by modifying an existing version [8] [9].

IJSER

1.2 Motivation

Regression test suites have been studied over various aspects such as reduction, prioritization, pruning and augmentation for decades. However, very few attempts have been made to design or re-engineer regression test suites for large miltiparameter software using methodical approaches. There exists a way to re-engineer or design the regression test suites using a Combinatorial approach. The tools provided by National Institute of Standards and Technology are very useful for this. In this Thesis the Contiki Operating System being developed for the Internet of Things is used as case study.

Contiki Oprating System has a regression test suite which is made publically available for study. Using this test suite and its measured bench marked coverage data as a base test suite, a study can be done on any re-engineered or fresh test suite. This can help in ascertaing whether the new test suite is indeed effective.

IJSER © 2019
http://www.ijser.org

Automation of the functional test script generation can make the process of executing a test suite easier. This has not been done for the Contiki Operating System. The test design tools output generic test design, the mapping of generic test case design to actual functional test cases needs to be done.

When the test suite is designed from scratch, the input parameter model refining can be concluded at the required level of code coverage. It is possible to use statistical techniques to develop a mechanism for the same. This Thesis addresses the motivations explained above and documents the same.

1.3 Problem statement

This Thesis studies the application of combinatorial testing to regression test suite of operating system used for internet of things. Following problem statements are addressed in the Thesis.

Objective 1: Study of Regression test suites in general

IJSER

Before studying the regression test suites of operating systems used for internet of things, generic study of regression test suites should be done. Important problems like regression execution time analysis and regression test suite prioritization using well defined algorithms are to be explored.

Objective 2: Design and implementation of combinatorial testing based test suites for internet of things operating system and its simulators

The combinatorial testing should be applied to regression test suite of one of the open source operating system such as Contiki and its simulator Cooja to either re-engineer the existing test suite or for freshly designing the regression test suite

IJSER © 2019
http://www.ijser.org

Objective 3: Measuring the effectiveness of designed test suites using the traditional coverage techniques like code coverage

The effectiveness of the re-engineered test suite or freshly designed test suite should be ascertained using the traditional metrices such as code coverage. The coverage data confirms the effectiveness of combinatorial test suite.

Objective 4: Automation of test scripts generation from combinatorial testing design model and analyzing coverage to refine the combinatorial testing design model

Where-ever possible automation of test script generation should be explored for combinatorial testing. The repetitive error prone manual test scripting should be replaced with the automation. The analysis of the code coverage should be done and the input parameters modeling should be refined iteratively to achieve the desired coverage in internet of things operating system or its simulator.

IJSER

Objective 5: Propose an integrated test environment for combinatorial testing

An integrated test environment that uses the combinatorial testing should be proposed. The proposal should have the tools and techniques that should be used for specific blocks of the integrated test environment. Both commercial and open source tools should be explored for the objective 5.

1.4 Research goals

This research focuses on application of combinatorial testing to internet of things operating system and its simulator. Following are the goals of research.

Research goal 1:

Analysis of execution time of regression test suite.

IJSER © 2019
http://www.ijser.org

Research goal 2:

Re-engineering the regression test suite or fresh design and implementation of combinatorial testing based regression test suite for Contiki and its simulator Cooja.

Research goal 3:

Quantification of effectiveness of the re-engineered or freshly designed regression test suite using the metrics code coverage.

Research goal 4:

Exploration of automation of test script generation from generic test design output generated using the combinatorial testing and mapping to actual functional test cases. Usage of code coverage data to refine the input parameter modeling in iterative manner.

Research goal 5:

Integrated test environment should be proposed for combinatorial testing.

IJSER

Research goal 6:

Study atleast one regression test suite prioritization algorithm.

1.5 Solution Approach

Before we provide details of our approach, we list the requirements for an ideal Regression test suite for an IoT OS. One of the desirable properties of a Regression test suite is to have execution time which is small. This is around twelve hours or overnight typically for a software under development. Another desirable property is to have maximum code coverage.

We propose a comprehensive methodology to automate generation of a functional Regression test suite. We model the effectiveness of the Regression test suite by using code coverage as the measure. This ability has enabled us

IJSER © 2019
http://www.ijser.org

to apply our methodology and study three large multiparameter software. These are software which have been made freely available for research.

1.6 Publications

PAPER A: "Regression Test Suite Execution Time Analysis using Statistical Techniques" published in, International Journal of Education and Management Engineering(IJEME),2016, Vol.6, No.3, pp.33-41, 2016.DOI: 10.5815/ijeme.2016.03.04

PAPER B: "Integrated test environment for combinatorial testing"published in Advance Computing Conference (IACC), 2015 IEEE International,2015,doi: 10.1109/IADCC.2015.7154802

PAPER C: "Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach" published in ACM SIGSOFT SEN, 2015,Volume 40 Issue 2,pp 1-3,doi:10.1145/2735399.2735413

PAPER D: "Test Suite Design Methodology Using Combinatorial Approach for Internet of Things Operating Systems" published in Journal of Software Engineering and Applications,2015, 8, 303-312. doi: 10.4236/jsea.2015.87031

IJSER

PAPER E: “Generating Effective Test Suite for Multiparameter Software using ACTS Tool and its Verification using Code Coverage Tools”, 2018, IJSER, Volume 9, Issue 8.

PAPER F: "An Attempt to Design and Implement Contiki and Cooja Regression Test Suites by Using Combinatorial Testing”, Published in IJSER, Jan 2019.

PAPER G: “Functional Test Case Generator for Contiki and Cooja”, Published in IJSER, Jan 2019.

PAPER H: "Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques" published in, International Journal of Education and Management Engineering(IJEME),2016,Vol.6, No.5, pp.32-39, 2016.DOI: 10.5815/ijeme.2016.05.04

1.7 Research Contributions

This Thesis studies problem statements and research goals mentioned in the previous sections and the same are documented as research papers. The subsequent section, Thesis outline, elaborates more on this.

PAPER A: Analyzes the regression test suite execution time of generic regression test suite. This addresses the research goal 1 and forms the chapter 3 of the Thesis.

PAPER B: Proposes the integrated test environment of the combinatorial testing. This address the research goal 5 and forms the chapter 4 of the thesis

PAPER C: Demonstrates how the re-architecturing should be done for case study Operating System Contiki and its simulator Cooja. This addresses the research goal 2 and forms the chapter 7 of the thesis.

PAPER D: Demonstrates the combinatorial testing based methodology for IoT Opertaing systems is in general. It addresses the research goal 2 and forms the chapter 5.

PAPER E: Explains the effectiveness of CT on a multi parameter software. It addresses the research goal 2 and forms the chapter 6.

PAPER F: Explains the approach of redesigning the regression test suite of Contiki and Cooja simulator test suite using combinatorial testing approach. This addresses the research goal 2, 3 and 4. It forms the chapter 8 of the Thesis.

PAPER G: Explains in detail the function test case generator created for Contiki and Cooja. This address the goal 4 specifically and forms the chapter 10.

IJSER © 2019
http://www.ijser.org

PAPER H: Analyzes the test suite prioritization algorithm namely Residual test coverage algorithm and enhances the algorithm. This addresses the research goal 1 and forms the chapter 11 of the Thesis.

1.8 Thesis outline

The outline of the rest of Thesis is as follows:

Chapter 2 – Literature Survey

This chapter provides a survey of relevant work on regression test suite creation for multiparameter software. Related work from the following fields is reviewed: Combinatorial testing, code coverage, tools available for test suite creation using combinatorial testing and tools for code coverage measurement. After that, IoT operating system.

Introduction to Software Test Coverage and its relevance to the design of Regression test suites is reviewed.

IJSER

Chapter 3 - Regression Test Suite Execution Time Analysis using Statistical Techniques

In this chapter statistical parameters such as probability distribution function and correlation coefficient of regression test suite are studied. These statistical parameters will be of help while interpolating or extrapolating the test execution time of any test suite.

Chapter 4 - Integrated Test Environment for Combinatorial Testing

In this chapter an integrated test environment for combinatorial testing is explored. Integrated test environment approach brings better integration of different tools for CT. It is a centralized approach which reduces the number of tools and duplicated functionality.

Chapter 5 - CT-RTS: Advanced Combinatorial Testing for Software Regression Testing

This chapter explains CT-RTS approach. This chapter discusses CT-RTS approach where the effectiveness of the test design generated from the ACTS output is ascertained using the code coverage tools with respect to traditional coverage metrices.

Chapter 6 –CT-RTS: Generating Effective Test Suite for Multiparameter Software using ACTS Tool and its Verification using Code Coverage Tools

This chapter explains how ACTS tool can be used to generate effective test suite for multiparameter software. The effectiveness of the CT-RTS generated test suite is cross verified using the traditional code coverage metrics.

Chapter 7 - Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach

This chapter proposes the re-architecture of Contiki and Cooja Regression test suites using combinatorial testing approach. Contiki testing environment, combinatorial testing, code cover tool usage and initial results of code coverage of base Contiki and Cooja regression test suites are discussed.

Chapter 8 - Test Suite Design Methodology using Combinatorial Approach for Internet of Things Operating Systems

In this chapter proposes test suite design methodology using combinatorial testing approach for internet of things operating systems. The typical work flow of base-lining the test suites is discussed to begin with. This chapter then discusses the redesigning the test suite if it already exists for the cases of inadequate coverage. It further discusses the case when the regression test suite is missing. The chapter also discusses Contiki specific environment changes needed for gathering the coverage data.

IJSER © 2019
http://www.ijser.org

Chapter 9 – CT-RTS: Contiki and Cooja Regression Test Suites Design and Implementation using Combinatorial Testing

In this chapter design and implementation of combinatorial testing based test suites for case study operating system Contiki and its simulator Cooja is studied. Chapter starts with the base regression test suite and then introduces the re-engineered test suite and the Cooja test suite which is created from scratch. The process of re-engineering is discussed in detail. The process of auto generation of the test cases is discussed in detail.

Chapter 10- CT-RTS: Combinatorial Testing based Regression Test Suite: Functional Test Case Generator for Contiki and Cooja

This chapter discusses the requirements, high level design, software implementation and usage details of the Functional test case generator for Contiki and Cooja.

Chapter 11 - Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques

In this chapter we study regression test suite prioritization using the residual test coverage algorithm. Our proposed algorithm is presented in this chapter. Our algorithm uses the test coverage and test execution time as parameters for breaking any tie which occurs among the candidate test cases for execution. Our approach helps in prioritizing the test cases when the number of test cases is very large. A black box approach to test suite prioritization is also presented in this chapter.

Chapter 12 – Conclusion

This chapter concludes the Thesis. It maps the research problems and research goals to individual chapters of the Thesis.

IJSER © 2019
http://www.ijser.org

2 Literature Survey

Literature survey contains six sections.

Properties and characteristic of IoT operating systems are studied to begin with. Then study of the importance of regression testing is explored. Study of the combinatorial technique based approach to software testing is explored. Study of the need for combinatorial testing for IoT Operating System is done. The tools which can be used for combinatorial testing are studied. The last part is software test coverage and its relevance to the design of regression test suites.

2.1 Properties and Characteristics of IoT Operating Systems

IJSER

Significant research effort in the field of IoT Operating Systems has been directed towards creating software with minimum device resource requirement. This is particularly important as these OS will in future be used in diverse devices . Some popular IoT OS and their details are listed below.

Contiki is an open source, portable, multi-tasking operating system for memory-constrained networked embedded systems developed by Adam Dunkels at the Networked Embedded Systems group at the Swedish Institute of Computer Science [2]. Contiki is designed for embedded systems with small amounts of memory. A typical Contiki configuration is 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki consists of an event-driven kernel on top of which application programs are dynamically loaded and unloaded at runtime. Contiki processes use light-weight protothreads that provide a linear, thread-like programming style on top of the event-driven kernel. Contiki also supports per-process optional preemptive multi-threading, interprocess communication using message passing through events, as well as an optional GUI subsystem with either direct graphic support or locally connected terminals or networked virtual display with VNC or over Telnet. Contiki

IJSER © 2019
http://www.ijser.org

contains two communication stacks:uIP and Rime. uIP is a small RFC-compliant TCP/IP stack that makes it possible for Contiki to communicate over the Internet. Rime is a lightweight communication stack designed for low-power radios. Rime provides a wide range of communication primitives, from best-effort local area broadcast, to reliable multi-hop bulk data flooding. Contiki runs on a variety of platforms ranging from embedded microcontrollers such as the MSP430 and the AVR to PC. Code footprint is of the order of kilobytes and memory usage can be configured to be as low as tens of bytes. Contiki is written in the C programming language. Contiki is freely available as open source code under a BSD-style license.

TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters [10]. Worldwide community from academia and industry use, develop, and support this operating system as well as its associated tools [11]. TinyOS is a tiny (fewer than 400 bytes), flexible operating system built from a set of reusable components that are assembled into an application specific system. TinyOS supports an event-driven concurrency model based on split-phase interfaces, asynchronous events, and deferred computation called tasks. TinyOS has been under development for several years and is currently in its third generation involving several iterations of hardware, radio stacks, and programming tools. Over one hundred groups worldwide use it, including several companies within their products.

IJSER

RIOT is an operating system designed for the particular requirements of Internet of Things (IoT) scenarios. These requirements comprise a low memory footprint, high energy efficiency, real-time capabilities, a modular and configurable communication stack, and support for a wide range of low-power devices. RIOT provides a microkernel, utilities like cryptographic libraries, data structures (bloom filters, hash tables, priority queues), or a shell, different network stacks, and support for various microcontrollers, radio drivers, sensors, and configurations for entire platforms, e.g. TelosB or STM32 Discovery Boards [12].

2.2 Importance of Regression Testing

Developers have to ensure that the system that is being developed is stable. This is especially true for the networking part as any problems in this could cause the entire system to collapse. Developers of the Contiki OS have made a lot of effort to make the system really stable, particularly the wireless IPv6 mesh networking. They have set up a regression testing framework that kicks in on every commit and runs the system on 9 different emulated hardware platforms with 4 different CPUs and on more than 1000 emulated wireless network nodes. This has resulted in the finding of several hard-to-find bugs deep in the network stack that show up as the envelope on system performance is pushed up. Several parts of the IPv6 stack has also been rewritten to make the code easier to follow and for everything to work better.

2.3 Combinatorial techniques based approaches to Software Testing

National Institute of Standards and Technology (NIST) published a widely cited report in 2003, which estimated that the inadequate testing costs the US economy dollar 59.5 billion every year, even though significant development budget goes towards testing in a typical software development cycle. The estimate of the same figure for all the software developed across the globe will be alarming. In this context, testing becomes important as a research topic. Various methodologies exist for testing any given product. NIST has pioneered the field of testing called combinatorial testing. The experience of NIST is documented in combinatorial testing manual publicly available on NIST website [13]. The latest survey [14] summarizes the interest and usage of Combinatorial testing by the Industry and Research communities.

2.4 The need of Combinatorial based testing techniques for IoT OS

There is a renewed emphasis on design of robust IoT OS to ensure that the diverse hardware and applications based on them run smoothly. Regression test suites and their redesign using Combinatorial techniques provide a good

IJSER

IJSER © 2019
http://www.ijser.org

first step in making IoT OS stable. Study of other aspects of testing and looking at issues such as code coverage done during the testing process is what will lead us to better designed test suites. Researchers have studied varous aspects of testing. These attempts are far from complete in the face of continuous growth in complexity and variety of devices and applications. There is a need to study these topics together for a better understanding of test suite development for IoT OS.

2.5 Use of Combinatorial technique based tools

A variety of software tools are available to assist with combinatorial testing projects. Various methodologies of Combinatorial testing exist in practice namely configuration based testing, input variable method of testing [15] and model based testing [16]. P.V Paolo Arcaini et. al. have done study on how testing can be applied to constrained environments [17]. A study of an industrial problem has been done by M. F Johansen et al [18]. Combinatorial testing has been applied to industry suite with success [15]

IJSER

Here we summarize a set of tools made available by the NIST ACTS project.

- ACTS covering array generator produces compact arrays that will cover 2-way through 6-way combinations [19].

- Coverage measurement tool produces a comprehensive set of data on the combinatorial coverage of an existing set of tests [20].

The ACTS covering array generator is faster and produces smaller test arrays than others, based on comparisons done in 2009. The ACTS tool itself has been tested using ACTS [21]. Raghu Kacker et al have presented papers in leading forums for CCM tool [22] [23].

.

IJSER © 2019
http://www.ijser.org

2.6 Software Test Coverage and its relevance to the design of Regression test suites

Code coverage is one of the quantitative measures to judge if the execution of a test set has been adequate. A variety of measures have been developed to gauge the degree of test coverage. Some of the coverage metrics are: Statement coverage: This gives the quantitative measure of how many statements are covered and how many statements are missed as part of coverage activity. Decision or branch coverage: This gives the quantitative measure of how many branches of each control structure are covered. Condition coverage: This is also called predicate coverage. The condition coverage gives the quantitative measurement of how many Boolean expressions in the code are evaluated to both true and false.

2.6.1 CodeCover

CodeCover is an open source tool meant for gathering the coverage data. It is developed in 2007 at University of Stuttgart, Germany. Code cover gives the statement coverage, branch coverage and condition coverages. Code cover supports the languages such as Java and Cobol although it can be extended for any programming language. Codecover supports the platforms such as Linux, windows and Mac OS. The tool can be used in command line, Ant integration or Eclipse mode.

2.6.2 OpenClover

OpenClover is open source tool for code coverage available since April 2017. OpenClover is platform independent making it suitable for operating systems such as Linux, Windows and MacOS. Clover combines the coverage and metrices to give the Total Percentage of Coverage (TPC) which highlights the risky code. OpenClover can be used in command line, Ant integration or Eclipse mode.

Clover and CodeCover are tools which have been used for measuring code coverage in Java code [24] [25].

IJSER © 2019
http://www.ijser.org

IJSER

2.7 Gaps in Existing Research

Key open research issues include the following:

- Regression test suites are designed to gain confidence in the testing process. A regression test suite ensures that there are no unintended side effects of code changes. A comprehensive Regression test suite would include exhaustive tests which cover every scenario possible. This also means impractical size and execution time. Research has been already done to keep the Regression test suite size practical [23]. Regression Test Selection (RTS) has been studied exhaustively [26]. Efforts have been done at NIST [27] to demonstrate that the test design if done using Combinatorial test design tools such as ACTS, it is possible to optimize the test suite while retaining effectiveness. Use of Combinatorial approach in redesigning the test suites for IoT Operating Systems when the test suite already exists has not been done earlier. Redesigning the Regression test suite of IoT OS such as Contiki has been done as part of this thesis.

- No study has been done on how to design the test suite for the IoT Operating Systems using Combinatorial approach when the Regression test suite does not exist. Operating Systems such as RIOT, TinyOs, Arch Linux do not have Regression test suite made available publically. In this thesis we develop a methodology which can be used to design a Regression test suite using our ACTS-RT approach.

- There is no systematic study which shows the effect of systematic Combinatorial testing approach over non combinatorial testing process. In this work we propose a formal testing process: generation of a test design document and mapping of test cases to actual functional test case. The test suite designed thus will be used to measure the effectiveness in terms of size and effectiveness. Effectiveness parameter used in this work is code coverage.

The objectives of our proposed research are to:

Objective 1: Study of regression test suites of operating systems used for internet of things operating systems. Characterize them using execution time analysis and code coverage.

IJSER © 2019
http://www.ijser.org

Objective 2: Develop a methodology for creation of combinatorial testing based regression test suite. The generated test suite can be applied to either re-engineer the existing test suite or for freshly designing the regression test suite. The effectiveness of designed test suites can be measured using code coverage

Objective 3: Demonstrate the methodology by applying it to multiparameter software. This will study the reaearch domain of Internet of Things operating systems.

Objective 4: Automation of functional test scripts generation from combinatorial testing design model and analyzing coverage to refine the combinatorial testing design model. Propose an integrated test environment for multiparameter software.

IJSER

IJSER © 2019
http://www.ijser.org

3 Regression Test Suite Execution Time Analysis using Statistical Techniques

3.1 Introduction

In this chapter we detail our work on the execution time analysis using statistical techniques. We start by discussing the need for studying the regression test execution time and then move on to sections which discuss how the execution time analysis can be performed.

When there is scarcity of resources, test suite execution time reduction is important. After generating test design using combinatorial approach and after applying test case selection, test suite minimization and prioritization, further the test execution time reduction needs to be investigated. Statistical techniques are effective in analyzing and reduction of the test execution time. When there is a need to augment the test suite, statistical techniques help in estimation of the execution time using extrapolation. Statistical techniques can also aid in choosing the best test setup in terms of Operating System, tools and Java virtual machine combination for a Java based test setup. Statistical techniques are one-time activies and the results are valid unless there is change in one of the layers of the test setup. Activities detailed in this chapter are carried out during the test setup planning and maintenance phase.

In this chapter we discuss about how statistical techniques can be applied to further analyze and reduce the test execution time at the test case level and test suite where possible. It may appear that the techniques themselves can be an overhead. But these activities will not be carried out during each execution of the test case. These are often upfront activities which the test team can carry out so that they have best hardware, best Java Virtual Macine (JVM) and best OS combination.

IJSER © 2019
http://www.ijser.org

3.2 Functional Simulator Tools

Functional simulator tools are used in both wired and wireless networks to test the product code. Test teams use automation and simulator tools often together. For many IoT Operating Systems there is a parallel development of simulator tools. Some of the advantages of having a simulator are:

- Test setup hardware costs can be high.
- The hardware availability may lag the software development cycle.
- The test up is scalable when simulator tools are employed.
- Product code debugging becomes easy with the simulators.
- It may not be possible to simulate some scenarios using hardware in real setup.
- Test setup using simulator is easy which makes test setup more flexible.

The above mentioned advantages list is brief. The simulator running on the host makes the product code believe as if it is running in real environment. The product code often resides on the same machine if it is host based testing and runs on a different machine if it is distributed testing. In distributed environment the connection can be TCP/IP based or the operating system will be distributed. Figure 1 shows the generic test setup.

IJSER

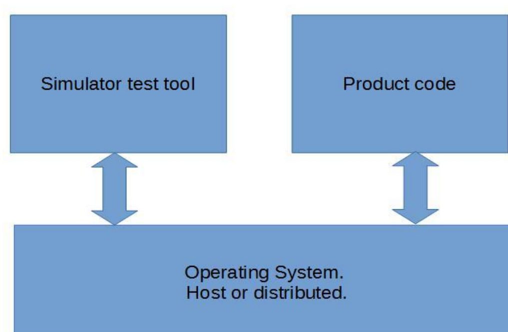


Figure 1. Generic Test Setup Involving Simulator Tools in Network

IJSER © 2019
http://www.ijser.org

3.3 Java Functional Simulator Tools

For a Java based System Under Test (SUT), Figure 2 gives a possible setup of the system from the tool's side. One of the aims of analysis is to study the test execution time and find mechanisms to reduce the execution time.

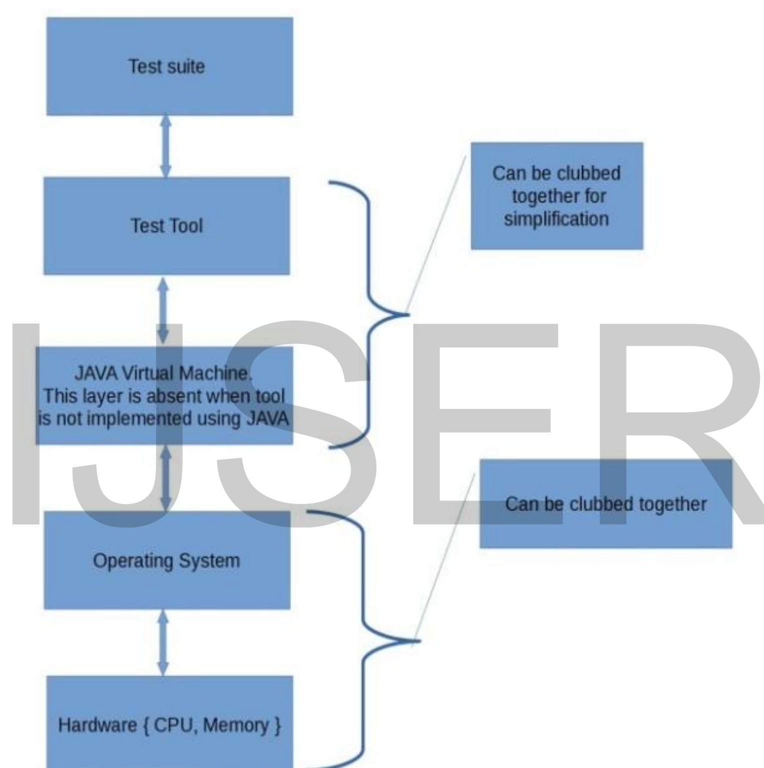


Figure 2. Generic Framework of Simulator Tools Explained.

As can be seen the Figure 2, there are multiple layers in the tools side. The test execution time is function of the execution in each layer. The total execution time of the test case will be a function of all three execution times f(x ,y ,z), here x is the time to execute the code of the test case. The component y is due to the time required for test tool and z is the time required for execution on the hardware. In several tools, these components are plug-and-play. For example, a version of the test suite with the particular version of the tool can run on various versions of the JVM(Such as Java 1.5,1.6,1.7 or 1.8) and various

IJSER © 2019
http://www.ijser.org

versions of the OS(Ubuntu, RHEL or any other flavor of Linux). Statistical techniques can be employed to choose various combination of the layers. For a given version of the JVM, it is possible to employ hotspots and fine tune the tool.

3.4 Java Hotspot VM Options

Java gives the options to fine tune the JVM. The broad categories are as follows.

- Behavioral options of the virtual machine.
- Garbage collection options.
- Virtual machine fine tuning options.
- Debugging options.

The above mentioned options will make the JVM behave in a particular manner depending upon the command line options. A set of command line options can be chosen and then employed for further evaluation of performance.

3.5 Test Case and Test Execution Time Observations

IJSER

Test case executed on particular setup of: tool, JVM, OS and hardware, will have different execution time each time it is executed. The JVM and OS introduce randomness in the test execution time. This implies that a straightforward prediction and calculation of test execution time is not possible. There is a need to develop Statistical techniques which can help in estimation of execution time.

3.6 Statistical Techniques for Execution Time Analysis

In this section we discuss applied statistical techniques. If there are three versions/variations of OS viz. OS1, OS2, and OS3 and three JVMs viz. JVM1, JVM2 and JVM3 and we want to compare, the table could look like as follows.

Table 1. Execution time of the test suite for various JVMs and OSs

Total test suite execution time JVM1	JVM2	JVM3
OS1		

IJSER © 2019
http://www.ijser.org

OS2		
OS3		

This is the coarse level analysis to come up with the best {JVM, OS} combination for a given test suite and hardware.

Since the given same test case gives different test execution time during different trials, it is best to execute the test suite multiple times and come up with the mean and standard deviations of the consolidated observations.

Now let us look at the observations at test suite level. If the best test case finishes the execution in 20 units of time and worst test case takes 80 units of time and if the test suite has certain mean and standard deviation we could plot the same using any mathematical tool. The graph could look like as in Figure 3. The Gaussian curve or normal curve can be explained as follows. The execution time of the test case in a given test suite is random variable taking continuous values in the given range. Further for very large test suite this will be smooth curve with inverted bell shape.

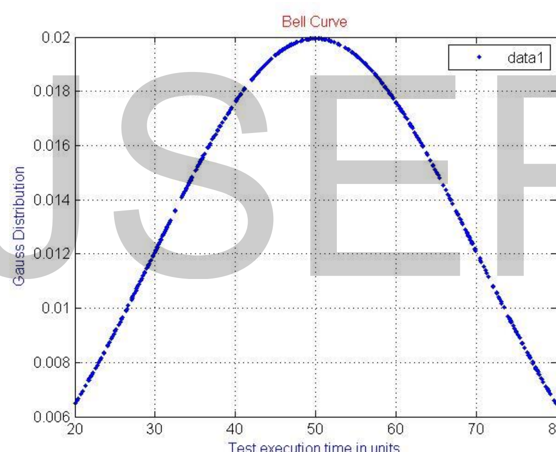


Figure 3. Normal Distribution Curve of Hypothetical Test Suite

If we plot the same curve for two different cases of table 1, the Figure could look as in Figure 4. Let us compare the two graphs in the Figure 4.

IJSER © 2019
http://www.ijser.org

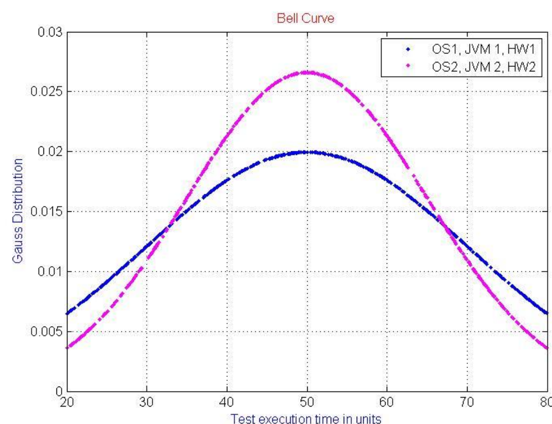


Figure 4. Normal Distribution Curves for the Same Test Suite on Two Different Setups.

These two figures are drawn with the help of MATLAB for hypothetical test setups with same mean execution time but with different standard deviation. As can be seen from the figure, the {OS2, JVM2, HW2} is more predictable with respect to test execution time. Further, the shape of the curve is Gaussian since the execution time due to OS and JVM makes the test execution time for a given test case random. This random value takes the continuous values and not discrete values leading to continuous distribution. In summary, we prefer that {OS, JVM, HW} combination which has least mean and least standard deviation.

Once we deduce the particular combination {OS, JVM, HW} as best combination, we could check the test suite execution time with various sets of command line options. Simply put choose the best set of command line options for a given {OS, JVM, HW} combination.

Coefficient of correlation between various parameters of a given command line option with test suite execution time tells which particular parameter within the given set of command line is influencing the execution time.

Numeric options are set with -XX:<option>=<number>. Numbers can include 'm' or 'M' for megabytes, 'k' or 'K' for kilobytes, and 'g' or 'G' for gigabytes (for example, 32k is the same as 32768).

IJSER © 2019
http://www.ijser.org

Table 2. Tabulation table for co-efficients of correlation

SI No	-XX Numeric option value		Test case execution time				
		X= x-$\bar{x}$		Y=y -$\bar{y}$	X^2	Y^2	XY
1							
2							
. . .							
N							

$$\text{Coefficient of correlation} = \frac{\sum XY}{\sqrt{\sum X^2 Y^2}} \quad (4)$$

This way of computing the coefficient of correlation may be tedious and further in JVM changing one parameter at a time may have some other side effects on total execution time of test case.

Going back to Gaussian curves, once we get the characteristic μ and σ of a given curve, i.e. characteristics of given setup for given test suite, the extrapolation of test suite time during test suite augmentation when the test cases get added or interpolation during pruning is not difficult.

IJSER

3.7 Limitations of Statistical Techniques

In some cases the tool, JVM and OS are tightly coupled for a given test setup. These techniques will not be of any use in such situations. For example, in the Contiki operating system used for IoT, the environment is Ubuntu like operating system. The Contiki in simulated environment comes with its own tool chain dependencies. Further the COOJA, official simulator of Contiki talks with Contiki using Java Native Interface (JNI). These are done with reason. The Contiki team wanted to have as user friendly environment as possible. However, when the tool setup employs plug and play architecture, the techniques can be of help.

When two setups are very different for example when the tool is migrated from old IRIX machines to latest RHEL machines, the latest machine will be

IJSER © 2019
http://www.ijser.org

clear winner because of the latest hardware, OS and JVM combination. There will be no need of these techniques.

Further, if the product code has internal timers based finite state machines, the approaches mentioned may not add much value.

3.8 Advantages of Statistical Approach

Test setups employing plug and play architecture and comprising of tools, OS and hardware can make use of the statistical techniques mentioned in this chapter. Test suites will have quantified statistical attributes with them making timing analysis simple. This makes the process of timing analysis convenient during the extrapolation and interpolation. The techniques can be employed during the augmentation and pruning of the test suites.

3.9 Conclusion

The statistical approaches mentioned in this chapter come with their own merits and drawbacks. As high-lighted in the draw back section, when the test setups use tightly coupled tools, OS and hardware cannot make use the techniques mentioned in this chapter. How ever, the test setups which employ plug and play architectures can immensely benefit with the statistical techniques mentioned in this chapter.

IJSER

IJSER © 2019
http://www.ijser.org

4 Integrated Test Environment for Combinatorial Testing

4.1 Introduction

In this chapter we present our proposed integrated test environment for combinatorial testing and its realization at a conceptual level. In this chapter we discus the entities that make up the integrated test environment and their interactions, highlighting the combinatorial aspects. Both manual and automated test environments can make use of the proposed environment. Test tools made available by researchers and commercial tool vendors need to be integrated to implement the environment.

A report published in 2003 by National Institute of Standards and Technology (NIST) estimates that the inadequate testing costs US economy $59.5 billion every year. If we sum up the costs due to improper testing of all the software developed across the globe, the cost will be alarming. This is the fact in spite of significant budget investment for software testing by typical software development cycle. In this light, new approaches are being used by the software professionals and researchers to reduce the cost due to poor quality of software. One of the possible approaches is combinatorial testing (CT). CT is pioneered by National Institute of Standards and Technology (NIST) [28]. NIST has created a bunch of tools which have been instrumental in adoption of CT. These tools can be combined with the commercial tools such as HP's quality center and research tools such as NuSMV to generate an integrated test environment for combinatorial testing.

An integrated test environment simplifies and streamlines the end to end testing process. The advantage of using an integrated test environment are many. It makes the creation of test cases, execution and evaluation simple without bringing up the individual tools separately.

The idea is to keep the integrated test environment as generic as possible while maintaining the interoperability between the tools. The individual entities are interconnected in the integrated test environment in a plug-and-play fashion.

IJSER © 2019
http://www.ijser.org

The steps involved in the realization of integrated test environment are as follows: search for commercially available tools and open source software tools, understanding the basic problems and approaches used for integrating the tools. Appropriate choice of the tools can vary depending upon nature and budget of the project. The current generation integrated test tool environment concepts are improvements over predecessors. Organizations are trying to automate the test process as far as possible since it eliminates the human error. Further, automation enables shorter for example nightly execution of the test suite.

While some of the previously published work on integrated test environment cater to specific domain (such as telecom), the concepts presented in this chapter are generic in nature which can be adopted for specific domain with minor tweaking. The core contribution of this chapter is proposing an integrated test environment, bringing together the different tools that are required for performing CT efficiently. In this environment some of the tools like defect analysis is CT specific, while tools for Test Management which are generic and well established, additional features that may be required from CT perspective are identified in this chapter. While the literature on integrated test environment is decade old, the integration of combinatorial aspect to integrated test environment has not yet been studied.

Integrated test environments have following benefits [29]:

- Better integration of different tools.
- Centralized approach.
- Less duplicated functionality.
- Reduced number of tools.
- Fewer dependencies.
- Simplified maintenance.

An integrated test environment can help immensely in the test driven development projects.

4.2 Overview of Integrated Test Environment

Figure 5. depicts a view of the integrated test environment being proposed in this thesis. The test model generator derives the test model from requirements and requirement changes. The output of the test model generator is the test model which is the input to the test generator. The test suite and test case generated by the test generator act as input to the test management tool.

IJSER © 2019
http://www.ijser.org

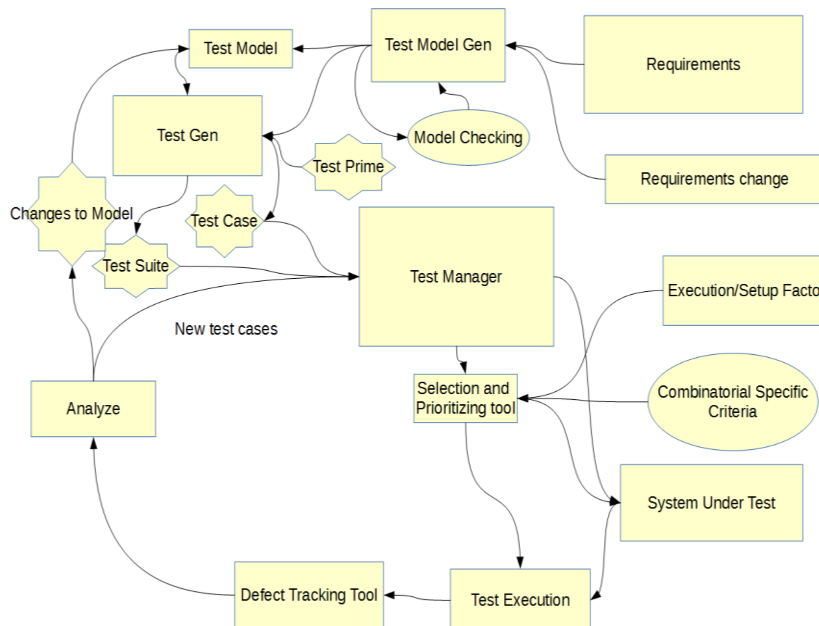


Figure 5 Integrated test environment entities.

IJSER

At the heart of the integrated test environment lies the test management tool (such as HP's quality center) whose inputs are the test cases and test suite generated by the test generators (such as Advanced Combinatorial Testing for Software). The test management tool drives the System Under Test (SUT). Selection and prioritizing tool prioritizes the test cases based on various parameters including the combinatorial specific criteria. These selected and prioritized test cases are executed to note down the bugs which are fed into the defect tracking tool. The defects are analyzed to suggest the changes in the model if need be. This could lead to new test model. Model checking tool (such as NuSMV) can be used to ascertain the correctness of the model.

4.3 Test Model Generator

Test model generator generates the test model from requirements and requirements change. The Combinatorial Test Design Model (CTD) model consists of the parameters, their values, and constraints existing among the various parameters and values. Model generation is the first phase of CT and

the model is the key input for test case generation. Hence deriving the combinatorial test design model is an important prerequisite for CT. In practical situations, the models are derived by test designers using their understanding of the software and their testing skills.

4.4 Test Generator

Test generator is that component of the integrated test environment which generates the test suite and test cases from the test model. For multiparameter software we find ACTS tool from the NIST can be used as the test generator.

ACTS covering array generator generates the compact array that will cover 2-way through 6 way combinations. Output of the ACTS tool is an excel spread sheet. The test management tool can read this excel sheet to populate the test case data base and create the required indices for test case selection based on combinations/interactions. There are other test generation tools whch are listed in the references.

IJSER

Table 3. Test generator tools

Product	Creator	License
ACTS	NIST	Free
Hexawise	Hexawise	Free and Commercial
Allpairs	Satisfice	Free, GPL

4.5 Test Management Tool

Test management tools are used to plan testing activities and report the status of quality assurance activities. Tools from different vendors have varying approaches to testing and thus have different set of features. These tools are used to maintain and plan manual testing, run or gather execution data from automated tests, manage multiple environments and to enter information about found defects. Test management tools offer the prospect of

IJSER © 2019
http://www.ijser.org

streamlining the testing process and allow quick access to data analysis, collaborative tools and easy communication across multiple project teams. Many test management tools incorporate requirements management capabilities to streamline test case design from the requirements. Tracking of defects and project tasks are done within one application to further simplify the testing. We list a few test management tools in Table 5.

Table 4. Test management tools

Product	***Creator***	***License***
HP's Quality Center	Hewlett packard	Proprietary
IBM Rational Quality Manager	IBM	Proprietary
Test Link	Team Test	GNU GPL

A study of these tools and usage reveal that for an ideal test suite, inclusion of some of the following combinatorial test specific features will be desirable:

- Being able to select testcase(s) covering a particular 2-way or higher order combination(s),
- Storing history of test effectiveness for different combinations used.
- Showing incremental combination coverage between selected test cases.

IJSER

4.6 Selection and Prioritization Tool

The objective of selecting the test cases for regression is to run a reduced and relevant set of test cases and the objective of prioritization is to run the selected test cases in the right order, so that bugs are captured early. For effective testing, prioritization is an essential activity, especially when there are a large number of test cases. The Test Selection and Prioritization tool can be augmented as a plug-in to the test management tool. The factors that can be considered for prioritization are combinatorial coverage criteria, bug history from previous runs, effort and time required for setting up and running the test cases. Coverage can be based on traditional coverage such as code coverage (such as CodeCover for Java projects) or more relevant combinatorial coverage using Combinatorial Coverage Measurement (CCM) tool for combinatorial testing. A cost based 2-way interaction coverage criterion has

been used to prioritize the test cases. Greedy algorithm generates prioritized test cases based on combinatorial coverage taking user specified weights into account. Regression test suites have been prioritized by combinatorial interactions among test cases. Output of the selection and prioritizing tool is an input for the test execution engine.

4.7 Defect Tracking Tool

The defect tracking tool or bug tracking tool is an application that can be used to track defects or bugs. The logging of the bugs can be done by the testers or by the end users, depending upon whether the defect was found during testing or use. There can also be a more formal process used by the organization to log the bugs on behalf of the end users. Defect tracking tool tracks the defects and its state from discovery till they get fixed, verified and closed. The defects raised in the defect tracking tool get assigned to the maintenance team for fixing. The assigned technical authority does a detailed analysis of the reported defect before working out the fix. Table 6 lists a few popular defect tracking tools.

IJSER

Table 5 Defect tracking tools

Client - Server	Open Source	Bug Zilla, Apache Blood hound
	Proprietary	Test track, FogBugz
Distributed	Fossil	
Hosted	Source Forge, Github, Google code	

4.8 Analysis

Input for analysis is the defect details from the defect tracking tool. The analysis is a complex activity and presently there are no automated tools available. The output of analysis should help in localizing the faults and working out a fix for them. In CT perspective, fault localization is essentially identifying the failure inducing combination (2 way / higher orders). There are systematic approaches reported in the literature like adaptive, non adaptive and machine learning methods. Delta debugging is an adaptive divide–and-conquer technique to locate interaction faults. A non-adaptive method proposed

IJSER © 2019
http://www.ijser.org

extends the covering array to the locating array to detect and locate interaction faults. Suspiciousness of tuple and suspiciousness of the environment of a tuple is considered to rank the possible tuples and generate the test configurations. A machine learning method to identify failure inducing combinations from a combinatorial testing constructs a classified tree to analyze the covering arrays and detect potential faulty combinations.

In general, all these methods require the tester to analyze and rerun some additional test cases to decide and localize the faulty combinations. For this task, the tester can use combinatorial criteria for further test selection. These features can be supported directly as a plugin into the test management tool or as part of the standalone selection tool.

4.9 Model Checking Tool

One of the most effective ways to produce test oracles is to use a model of the system under test, and generate complete tests, including both input data and expected results, directly from the model [27].Tools such as NuSMV (Nu Symbolic Model Verifier) can be used for this purpose. NuSMV was developed by Carnegie Mellon University, University of Genova and University of Trento. NuSMV can be installed on Unix/Linux or Windows systems. As long as the system has formal or semiformal specifications of the system under test, the NuSMV can be used.

4.10 Conclusion

In this chapter we presented generic frame work for an integrated test environment which can be adapted to specific domain with minor modification. The integration of combinatorial testing brings in the advantages of combinatorial testing to traditional testing approach. The concept and ideas presented in this chapter can be implemented to demonstrate a prototype -a proof of concept. Also the integrated test environment can be refined in a domain specific way such as domains where combinatorial testing is extensively used.

IJSER © 2019
http://www.ijser.org

5 CT-RTS: Combinatorial Testing based Software Regression Suite

5.1 Introduction

In this chapter we give the details of our proposed approach called Combinatorial Testing based Regression Test Suite (CT-RTS). The proposed approach, which can be used for multiparameter software, generates a regression test suite for a System Under Test (SUT) using the NIST-ACTS tool. The effectiveness of the generated test suite is ascertained using the code coverage tools.

Figure 6. depicts the process flow diagram for the CT-RTS. First the system under test needs to be studied before modeling the test input for the ACTS tool. The test model is essentially collection of parameters and parameter values along with the constraints and relations between the parameter values. The ACTS tool uses the test input or test model to generate the test design. The test design is essentially collection of rows. Each row is test case.

In few cases the test cases are readily executable as in case of Graphical User Interface SUTs. In other cases, the test cases are not readily executable. Therefore, intermediate steps are required to convert these ACTS generated test cases into executable test cases. In this Thesis, we refer these executable test cases as “functional test cases”.

The process of functional test case generation is tightly coupled with the target test environment. Simply put, the functional test case generation is case specific.

This Thesis applies the CT-RTS approach to two kinds of software

1. When the ACTS output test cases are readily executable
2. When the ACTS output test cases are converted to functional test cases.

IJSER © 2019
http://www.ijser.org

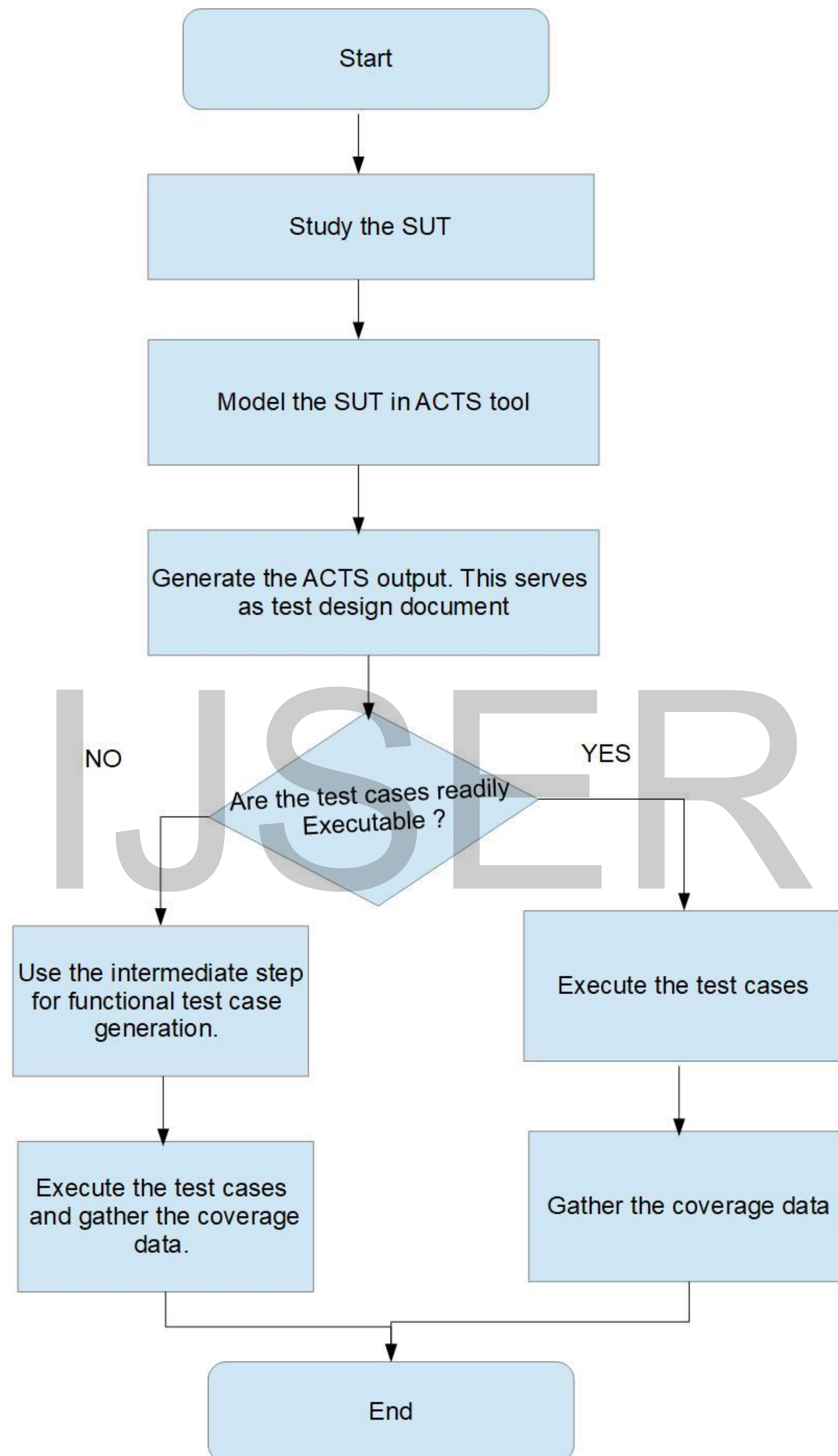


Figure 6. Process flow diagram for CT-RTS

IJSER © 2019
http://www.ijser.org

5.2 CT-RTS: Readily Executable Test Cases

As mentioned in the previous section, this is the case when the ACTS output test cases can be readily executed on the SUT. College Time Table which is a multi parameter software used as SUT and the findings are documented in the chapter 7 titled, “Generating Effective Test Suite for Multiparameter Software using ACTS Tool and its Verification using Code Coverage Tools”.

5.3 CT-RTS: Functional Test Case Generation

This is the case when the ACTS output test cases need to be converted to executable test cases using the intermediate step. The executable test cases are called functional test cases. IoT operating system Contiki and its Java simulator tool Cooja are used as the SUT and the findings are documented in the chapter 10 and chapter 11.

5.4 Conclusion

IJSER

This chapter introduces the CT-RTS approach where the ACTS generated test cases are executed on the target SUT and the effectiveness of the generated regression test suite is ascertained using the code coverage tools with respect to code coverage metrics. Two cases are discussed when the test cases are readily executable and the case when the intermediate step is required to generate the functional test cases. This chapter acts as the foundation for the chapter 7 and chapter 10.

IJSER © 2019
http://www.ijser.org

6 CT-RTS: Generating Regression Test Suite for Multiparameter Software and its Verification using Code Coverage Tools.

6.1 Introduction

Combinatorial testing is a practical method to test software with multiple input parameters. National Institute of Standards and Technology has developed tools which aid combinatorial testing. ACTS is one such tool which is freely available to users. In spite of this, very few software being developed are being tested systematically. In this chapter we explore the effectiveness and suitability of ACTS tool to test software which has a multiparameter input. We chose a Java based software, College Time Table, a software which involves multiparameter input, as system under test. We could achieve 90% coverage of instructions, line, method and 100% class coverage with practical time and effort with ACTS tool. The process involved in getting above mentioned results is documented in this chapter.
Empirical data generated with the code coverage confirms the effectiveness of ACTS generated test suite for a simple project. In a typical Software Development Life Cycle (SDLC) 50-80% budget goes towards testing. In spite of this, 2003 National Institute of Standards and Technology (NIST) report estimated a loss of 59.5 billion dollars to US economy due to inadequate testing [13]. In this light, software testing becomes critical. This is all the more critical as developers use pre-existing code which are now available as part of open source projects.

Several approaches exist to enable testing. One such field of testing is Combinatorial Testing (CT). A report by Nie indicates that CT is widely adopted by industry and researchers alike [14]. Exhaustive testing of all the possible combination of inputs and execution paths is a laborious task involving impractical man hours. CT is a method that can reduce cost and increase the

IJSER © 2019
http://www.ijser.org

effectiveness of testing. Pairwise testing is already in practice where 2-way combination of parameters is tested. However, as per NIST two way testing misses 10% to 40% or more system bugs [13]. Therefore higher level inter-action testing is critical. New algorithms have made 4 way to 6 way testing possible. Some of the rare combination of inputs trigger the failures that would have escaped previous testing or extensive use. Such failures are known as interaction failures. Traditional pairwise testing targets the 2-way interaction failures and has been in practice for quite some time. Till recently most tools would take impractically long time for generating the 3-way through 5-way arrays as the array generation process is mathematically complex. But the development of new algorithms recently has made the 3-way through 5-way array generation possible [3]. Two forms of combinatorial testing is possible: Configuration based combinatorial testing and Input parameter based combinatorial testing. A combination of both can also be used. Variety of software tools are available to assist CT. NIST has developed tools for this purpose. They are

1) Automated Combinatorial Testing of Software (ACTS).
2) Combinatorial Coverage Measurement (CCM) tool.
3) Sequence Covering Array Generator.

In this chapter we present our proposed approach called ACTS-RT. In the we take ACTS tool [19] and develop a mechanism for its usage and show its effectiveness for the purpose of CT. CCM tool is useful for measuring the combinatorial coverage [20]. Sequence covering array generator is useful when sequences are involved.

We successfully used our ACTS-RT on a Java based software, College Timetable, which is free and available on Source-Forge.

6.2 Brief Literature Survey of CT

Much work has been done on estimating the fault detection effectiveness of CT [22]. The basic assumption of the CT is faults are deterministic in that failure triggering combination of input values always produce failures if it is present in the input. There much study being done on the tools and usage of CCM [23]. M F Johansen has used CCM to test soft-ware of industrial size in his PhD Thesis [18]. M F Johansen describes product line testing, which is the strategic testing of the product line to gain confidence for any configuration of it. CT has also

IJSER © 2019
http://www.ijser.org

been applied to industrial test suites [30]. Laleh et al. talk about input space modeling methodology, before applying CT to a system, the input space must be modeled [15]. CT approach is unique in each paper and is evident from few papers [16], [17]. P. Amman et al. talk about how to combine model checkers with specification based mutation to generate the test cases from formal software specification. Paolo et al. talk about validation of model and test tool generated test suites [17].

6.3 ACTS Tool

ACTS tool is a test generation tool. It generates t-way combinatorial tests sets. The “t” in t-way can range from 1 through 6. A system in ACTS tool is specified by a set of parameters and their values. Given any t parameters (out of all the parameters) of system, every combination of values of these t parameters is covered by atleast one test in the test set.

ACTS makes use of several algorithms for the test generation. The algorithms include IPOG, IPOG-D, IPOG-F and IPOG-F2. IPOG-D is preferred for larger systems while IPOG, IPOG-F, IPOG-F2 work best for moderate size system.

IJSER

6.4 Open Clover

OpenClover is an open source code coverage tool [24]. Code coverage is a quantitative data about code covered as a part of test execution. It shows which part of the code is tested and which is not tested. Typically, tester does the code coverage in an iterative manner till the required criteria is met. Code coverage is done for the following reason:

- To know whether testing is adequate.
- To maintain the quality of code.

There exist three types of coverage tools. They are:

- Source code instrumentation tools
- Byte code instrumentation tools
- Runtime information collecting tools.

Clover uses the source code instrumenting as the source code instrumenting produces more accurate results. Types of coverage measured by various tools are:

- Statement coverage

IJSER © 2019
http://www.ijser.org

- Branch coverage
- Method coverage.

Clover combines the above mentioned criteria viz. statement coverage, branch coverage and method coverage criteria to arrive at Total Percentage of Coverage (TPC).
TPC is calculated as follows:

$$TPC = (BT + BF + SC + MC)/(2 \times B + S + M) \times 100\%$$

where

BT: Branches that evaluated to "true" at least once
BF: Branches that evaluated to "false" at least once
SC: Statements covered
MC: Methods entered

B: Total number of branches
S: Total number of statements
M: Total number of methods

Clover actually sees the real code structure and uses the source code instrumentation. Only line coverage is possible with byte code instrumentation tools. However, statement coverage is possible with Clover as it uses the source code instrumentation.

6.5 College Time Table

College Time Table (CTT) software is a simple Java based tool for generating small sized school or college time table [31]. CTT utility collects the information on the fly without expecting the details such as number of teachers, their name, subjects etc.
We have picked CTT for demonstrating the ACTS tool usage since it has a:
1) Practical sized code
2) Combinations of parameters are used as input and therefore it is a suitable software to demonstrate the functionality of ACTS.
3) We wanted to demonstrate the applicability of the method proposed by us on a software which we had not much detailed knowledge about.

IJSER © 2019
http://www.ijser.org

6.6 CT-RTS: Generating The Test Cases and Gathering Coverage Data

Table 6. Parameter and their values in ACTS for CTT software

Parameter	Parameter values
File_Operation	New_Time_Table, Save_Time_Table, Save_Time_Table_As, Load_Time_Table, Null
Print_Operation	Print_Current, Print_All_Individuals, Print_All_Classes, Print_Master_Table
Load_Demo_Time_Table	Demo_Time_Table, Null
Time_Table_Operation	Printer, Global_Counts, Remove_Gaps_Doubles, Freeze_Cell, Multi_Freeze, Clear_Freeze, Find, Next_Find, Find_And_Replace, Wizard-01, Insert_Row, Swap_Time_Table, Wizard-02, Delete_Row
Constraints: (File_Operation != "Null") => (Load_Demo_Time_Table == "Null") (Load_Demo_Time_Table != "Null") => (File_Operation == "Null")	

ACTS tool needed for this work was downloaded from the NIST website. CTT software which involves multiple input parameters and their combinations was taken from the github. The target SUT (CTT in this case) was imported in Integrated Development Environment (IDE) which was Eclipse. Once chosen SUT is imported, next step is installation of the code coverage plugin tool. OpenClover plugins are available for many popular IDEs. The SUT (CTT) is instrumented and built. Next step is to launch the SUT as usual.

IJSER

IJSER © 2019
http://www.ijser.org

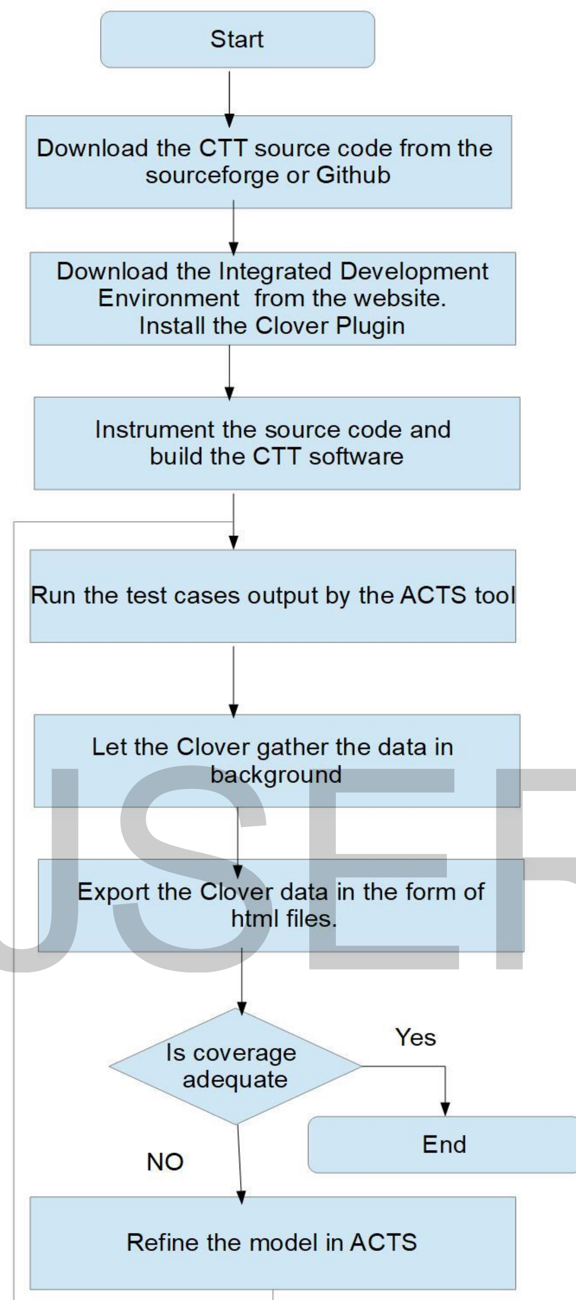


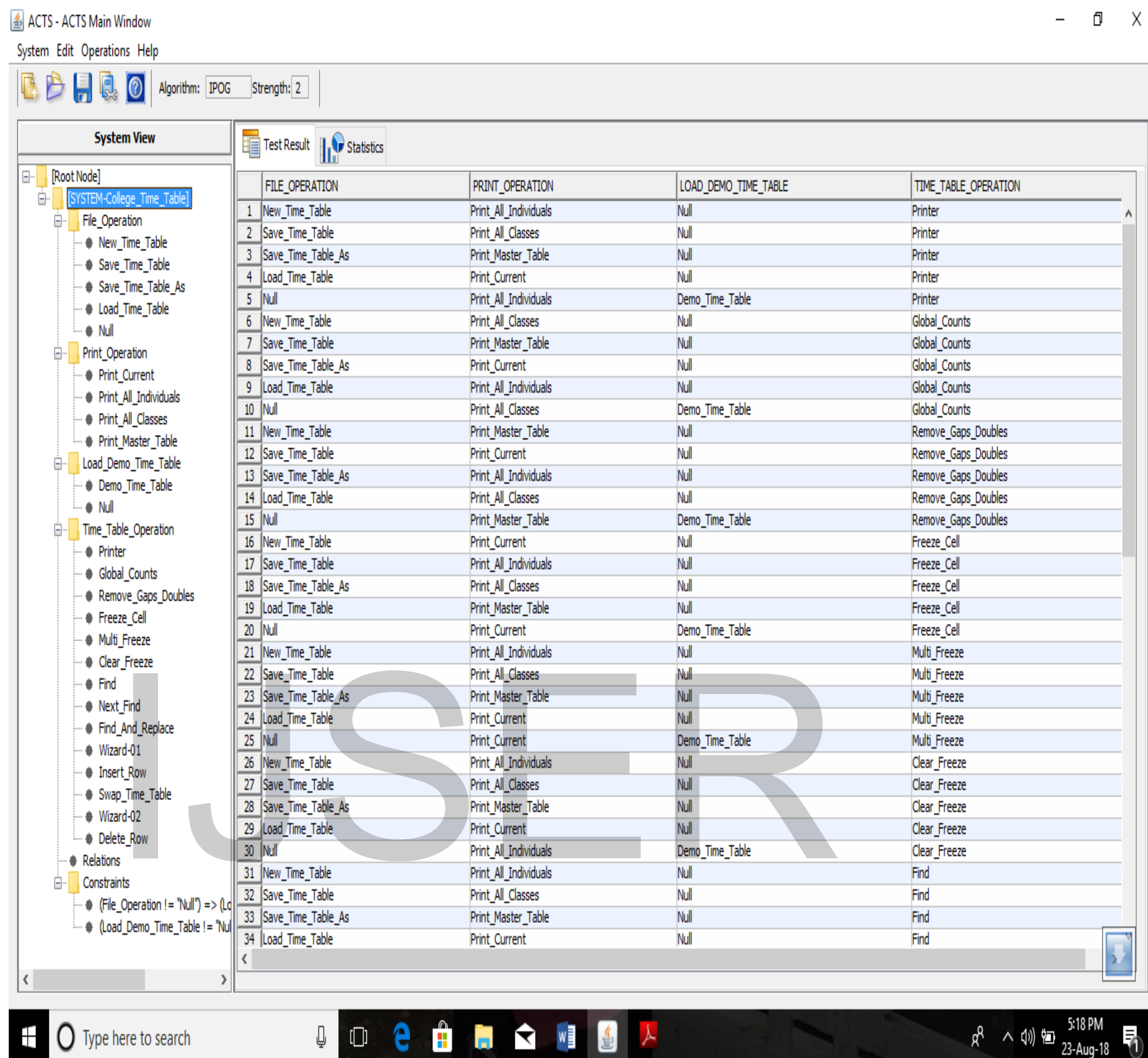
Figure 7. Process flow diagram for generating the ACTS test suite for CTT software and measuring the coverage

6.6.1 ACTS Tool Usage for Generating The Test Cases

ACTS tool comes with user guide. User guide contains information on how the tool needs to be used. ACTS tool was launched. The parameters and parameter

IJSER © 2019
http://www.ijser.org

values along with relations and constraints if any need to be populated in system under ACTS tool.



	FILE_OPERATION	PRINT_OPERATION	LOAD_DEMO_TIME_TABLE	TIME_TABLE_OPERATION
1	New_Time_Table	Print_All_Individuals	Null	Printer
2	Save_Time_Table	Print_All_Classes	Null	Printer
3	Save_Time_Table_As	Print_Master_Table	Null	Printer
4	Load_Time_Table	Print_Current	Null	Printer
5	Null	Print_All_Individuals	Demo_Time_Table	Printer
6	New_Time_Table	Print_All_Classes	Null	Global_Counts
7	Save_Time_Table	Print_Master_Table	Null	Global_Counts
8	Save_Time_Table_As	Print_Current	Null	Global_Counts
9	Load_Time_Table	Print_All_Individuals	Null	Global_Counts
10	Null	Print_All_Classes	Demo_Time_Table	Global_Counts
11	New_Time_Table	Print_Master_Table	Null	Remove_Gaps_Doubles
12	Save_Time_Table	Print_Current	Null	Remove_Gaps_Doubles
13	Save_Time_Table_As	Print_All_Individuals	Null	Remove_Gaps_Doubles
14	Load_Time_Table	Print_All_Classes	Null	Remove_Gaps_Doubles
15	Null	Print_Master_Table	Demo_Time_Table	Remove_Gaps_Doubles
16	New_Time_Table	Print_Current	Null	Freeze_Cell
17	Save_Time_Table	Print_All_Individuals	Null	Freeze_Cell
18	Save_Time_Table_As	Print_All_Classes	Null	Freeze_Cell
19	Load_Time_Table	Print_Master_Table	Null	Freeze_Cell
20	Null	Print_Current	Demo_Time_Table	Freeze_Cell
21	New_Time_Table	Print_All_Individuals	Null	Multi_Freeze
22	Save_Time_Table	Print_All_Classes	Null	Multi_Freeze
23	Save_Time_Table_As	Print_Master_Table	Null	Multi_Freeze
24	Load_Time_Table	Print_Current	Null	Multi_Freeze
25	Null	Print_Current	Demo_Time_Table	Multi_Freeze
26	New_Time_Table	Print_All_Individuals	Null	Clear_Freeze
27	Save_Time_Table	Print_All_Classes	Null	Clear_Freeze
28	Save_Time_Table_As	Print_Master_Table	Null	Clear_Freeze
29	Load_Time_Table	Print_Current	Null	Clear_Freeze
30	Null	Print_All_Individuals	Demo_Time_Table	Clear_Freeze
31	New_Time_Table	Print_All_Individuals	Null	Find
32	Save_Time_Table	Print_All_Classes	Null	Find
33	Save_Time_Table_As	Print_Master_Table	Null	Find
34	Load_Time_Table	Print_Current	Null	Find

Figure 8. ACTS tool populated data for CTT

Once these are populated the system can be built. Building the system is making ACTS tool generate the test cases. Each row of the output is one test case. To begin with certain combination of parameter and parameter values can be chosen along with constraints and relations. All the test cases in ACTS tool can be run. At the end of run of all the test cases, Clover would have collected the data in the background. The open Clover data can be exported in various output forms. In this case html format was chosen. From the output one can infer the data such as:

- Instruction coverage
- Branch coverage
- Condition coverage
- Line coverage
- Method coverage
- Class coverage

If the required criteria of code coverage is not met, the parameter modeling the ACTS tool can be revisited and refined. It is an iterative process and can be repeated till the required coverage criteria is met.

Figure 7. summarizes the steps mentioned for this work.

Table 6. gives the final set of parameter and parameter values along with constraints chosen for this work. The iterative work was concluded for 90% instruction coverage. Results section discusses the results in detail.

Although the steps mentioned in section 5 and subsection 5.1 are specific for a given software (CTT), for a given IDE (Eclipse) and code coverage tool (OpenClover), the steps involved are generic in nature and can be used for other soft-ware where combinations are involved.

Following configuration was chosen:

- Algorithm used: IPOG
- Strength chosen: 2
- Mode chosen: Scratch
- Constraint handling: CSP Solver

ACTS output statistics are as follows:

- Number of test cases: 71
- Number of covered combinations: 188

For the above mentioned configuration, ACTS took mere 0.094 seconds to generate the output to be exported in various formats.

6.7 Results and Results Analysis

Table 7 summarizes the results of test execution for CTT project.

Table 7. Clover coverage data for CTT software

CTT Project			
Entity	**Total**	**Missed**	**Covered**
Instructions	13483	1080	91%

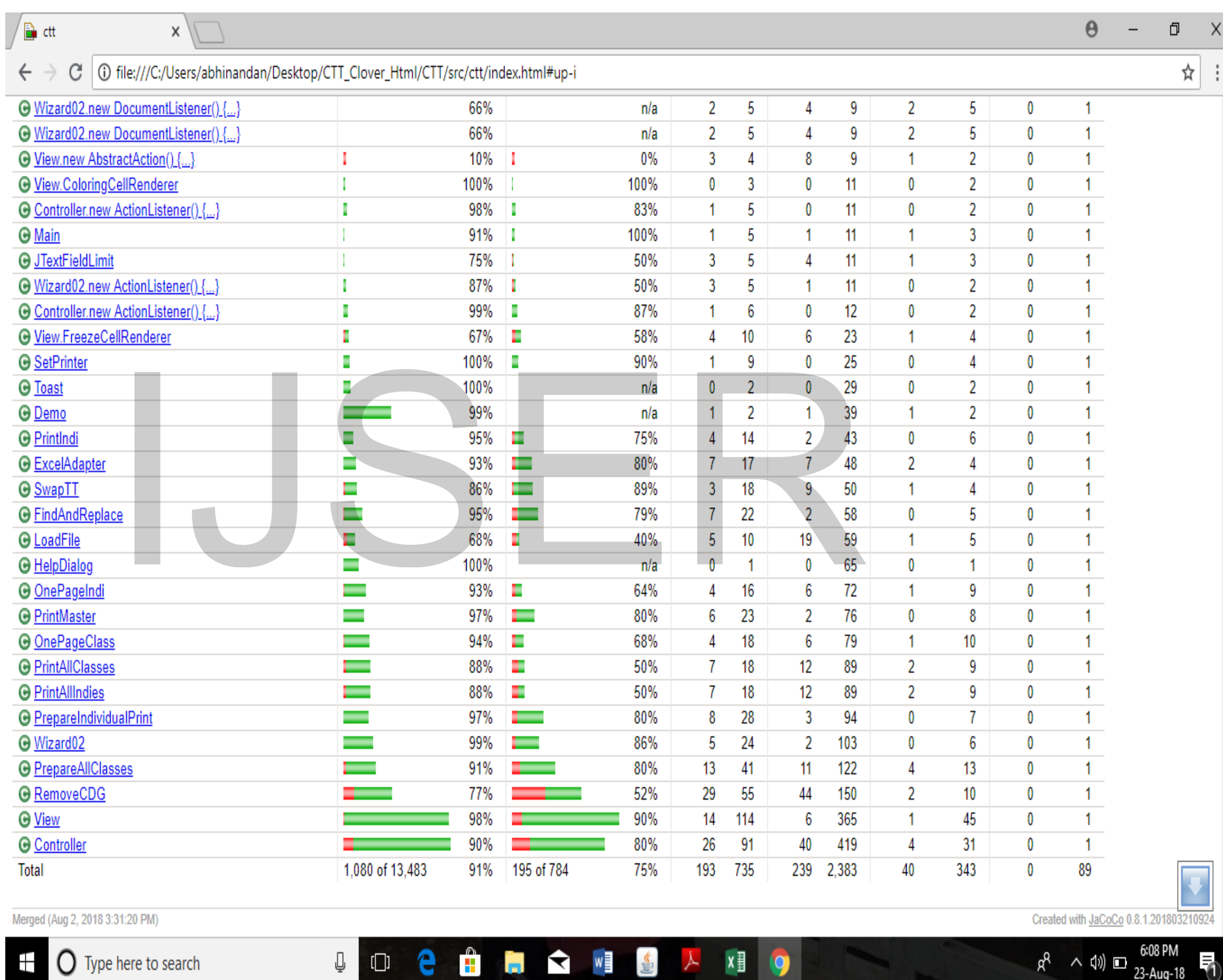
IJSER

Branches	784	195	75%
Lines	2383	239	90%
Methods	343	40	89%
Classes	89	0	100%

src

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ctt		91%		75%	193	735	239	2,383	40	343	0	89
Total	1,080 of 13,483	91%	195 of 784	75%	193	735	239	2,383	40	343	0	89

Figure 9. OpenClover output window at the project level



ctt
file:///C:/Users/abhinandan/Desktop/CTT_Clover_Html/CTT/src/ctt/index.html#up-i

Wizard02.new DocumentListener() {...}		66%		n/a	2	5	4	9	2	5	0	1
Wizard02.new DocumentListener() {...}		66%		n/a	2	5	4	9	2	5	0	1
View.new AbstractAction() {...}		10%		0%	3	4	8	9	1	2	0	1
View.ColoringCellRenderer		100%		100%	0	3	0	11	0	2	0	1
Controller.new ActionListener() {...}		98%		83%	1	5	0	11	0	2	0	1
Main		91%		100%	1	5	1	11	1	3	0	1
JTextFieldLimit		75%		50%	3	5	4	11	1	3	0	1
Wizard02.new ActionListener() {...}		87%		50%	3	5	1	11	0	2	0	1
Controller.new ActionListener() {...}		99%		87%	1	6	0	12	0	2	0	1
View.FreezeCellRenderer		67%		58%	4	10	6	23	1	4	0	1
SetPrinter		100%		90%	1	9	0	25	0	4	0	1
Toast		100%		n/a	0	2	0	29	0	2	0	1
Demo		99%		n/a	1	2	1	39	1	2	0	1
PrintIndi		95%		75%	4	14	2	43	0	6	0	1
ExcelAdapter		93%		80%	7	17	7	48	2	4	0	1
SwapTT		86%		89%	3	18	9	50	1	4	0	1
FindAndReplace		95%		79%	7	22	2	58	0	5	0	1
LoadFile		68%		40%	5	10	19	59	1	5	0	1
HelpDialog		100%		n/a	0	1	0	65	0	1	0	1
OnePageIndi		93%		64%	4	16	6	72	1	9	0	1
PrintMaster		97%		80%	6	23	2	76	0	8	0	1
OnePageClass		94%		68%	4	18	6	79	1	10	0	1
PrintAllClasses		88%		50%	7	18	12	89	2	9	0	1
PrintAllIndies		88%		50%	7	18	12	89	2	9	0	1
PrepareIndividualPrint		97%		80%	8	28	3	94	0	7	0	1
Wizard02		99%		86%	5	24	2	103	0	6	0	1
PrepareAllClasses		91%		80%	13	41	11	122	4	13	0	1
RemoveCDG		77%		52%	29	55	44	150	2	10	0	1
View		98%		90%	14	114	6	365	1	45	0	1
Controller		90%		80%	26	91	40	419	4	31	0	1
Total	1,080 of 13,483	91%	195 of 784	75%	193	735	239	2,383	40	343	0	89

Merged (Aug 2, 2018 3:31:20 PM)
Created with JaCoCo 0.8.1.201803210924
Type here to search
6:08 PM
23-Aug-18

IJSER

Figure 10. Open Clover output window granular level

Appendix B gives the visual output for the table. Detailed logs are kept in Google docs repository [32] and they act as supplementary information. As can be seen from the Table 2, class coverage was 100% while lines, instruction and method coverage was around 90%. Branch coverage was around 75% for the

70 test cases. Further, only success path in the code was tested and the error scenarios were not explored. In addition, some dead code existed in the CTT project. As mentioned earlier, since the vital entities were around 90% (Except branch coverage), the refining of parameter model in the ACTS was concluded.

6.8 Conclusion

In this chapter we present an implementation of use of combinatorial testing based tool for generation of test suite. We have proposed ACTS-RT a method for generating Regression test suite. The proposed method uses tools provided by NIST, ACTS and OpenClover. We use Clover a tool for for measuring the code coverage. We document the findings of generating test suite for multi parameter software and its verification using code coverage tools. We use as a SUT a multiparameter Java based soft-ware. We find that the implementation can be used on any mutiparameter software. The effectiveness of the ACTS tool generated test suite is cross verified with traditional metrics code coverage. The documented process in this chapter could achieve 90% coverage for the vital entities of coverage metrics with practical time and effort. The process documented in this chapter could be effectively used for any softwares which involve combinations of input parameters. Since several free software are used by users with their modifications incorporates, the method detailed in this work provides an effective way of testing.

IJSER

IJSER © 2019
http://www.ijser.org

7 Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach

7.1 Introduction

In this chapter, we describe how combinatorial testing can be applied to re-architecture Contiki and Cooja regression test suites. Contiki is the popular and widely accepted internet of things operating system. Combinatorial testing was pioneered by National Institute of Standards and Technology. National Institute of Standards and Technology offers a set of tools to public. One such tool is Automated Combinatorial Testing for Software. We describe how Automated Combinatorial Testing for Software can be used to generate a complete test suite for Contiki and Cooja. Coverage of base test suite is gathered using CodeCover, a code coverage tool for Java. The low percentage of coverage in Cooja indicated the need for a redesign of test suite. Once the base regression test suite is modified using Combinatorial Testing approach, it can be the new base regression test suite.

IJSER

Contiki is a popular internet of things operating system with a built-in Cooja (written in java) simulator. The latest Contiki operating system, 'version 2.7' has enhanced the regression test suite in latest version of Contiki. However, there is scope for improving the test suite further. In this regard, the test suite will be designed using Automated Combinatorial Testing for Software (ACTS). Each test case generated will be mapped to actual test cases to be executed in the Cooja environment of Contiki. Contiki supports various hardware platforms and a subset of those in simulation mode [33].

When the existing regression test suite was being studied it was observed that, the test cases are concentrated around a few motes and not evenly distributed across motes. If the test design is done using ACTS for the configuration testing as described in the text book and manual of combinatorial testing the test cases will be distributed evenly across the mote types (hardware).

IJSER © 2019
http://www.ijser.org

We instrumented Cooja using CodeCover to gather the coverage data with the base test suite. The coverage in Cooja, which is an indicator of the effectiveness of the test suite, was less than 20%. The low coverage however can be explained by the non-GUI (Graphical User Interface) mode of the Cooja in regression test case execution. Low percentage of coverage re-enforced the need for a different test strategy to test the Contiki operating system.

7.2 Contiki Testing Environment

Contiki gives an user friendly environment for testing the delta development on the operating system in the form of instant Contiki. The VMWare Player can be used to launch Ubuntu-like environment, which comes with the tool chain for developing the Contiki modules. Contiki can be built for multiple target platforms using the appropriate make file arguments. Contiki supports several mote types in simulation mode. The Cooja simulator talks to the compiled modules of Contiki using the Java Native Interface (JNI).

The existing regression test suite is comprised of many csc files in regression testing folder of Contiki. These are basically xml files understandable by Cooja. The csc files are designed for a given mote type or several mote types. Eighty three test cases of this type can be found in the regression test folder. Although the Cooja environment supports various hardware platforms in simulation mode the test cases are concentrated around a few mote types. This is not a good test design.

IJSER

7.3 Combinatorial Testing

If papers submitted in various forums are any indication, combinatorial testing has recently gained the acceptance among the researchers and industry as per Nie survey [14]. NIST has been supporting and guiding the combinatorial testing activity.

ACTS and CCM (Combinatorial Coverage Measurement) tools are used in our proposed method. ACTS makes use of various algorithms to generate the test suites for the end user. In our study covering array generated by ACTS served as the test case design document.

It is observed that the ACTS generated sequence array evenly distributes the test cases across mote types. Further, the following configuration is used as input to the ACTS tool.

IJSER © 2019
http://www.ijser.org

The table 8 gives the parameters and parameter values which are the inputs to the ACTS tool for Contiki regression test suites. The ACTS tool can be used in the GUI and non-GUI mode. Population of the parameter type and values in the ACTS tool is documented in the manual [2].The values could be populated in the GUI or can be supplied as text file.

The following configuration is used in the ACTS tool:

Degree of interaction coverage: 2 ,Number of parameters: 9, Maximum number of values per parameter: 10

Table 8. Input parameters for the ACTs tool

Parameters	Parameter values
Platform	Exp5438, z1, wismote, micaz, sky, jcreate,, sentilla-usb, esb, native, cooja
base	Multithreading, coffee, checkpointing
Rime	collect, rucb, deluge, runicast, trickle, mesh
NetPerformance	NetPerf, NetPerf-lpp, NetPerf-cxmac
collect	shell-collect, shell-collect-lossy
ipv4	telnet-ping, webserver
ipv6	ipv6-udp, udp-fragmentation, unicast-fragmentation, ipv6-rpl-collect
RPL	up-root, root-reboot, large-network, upanddownroutes, temporaryrootloss, randomrearrngement, rpl-dao
ipv6apps	servreg-hack, coap

7.4 CodeCover Tool Usage

The paper on CodeCover [25] explains the versatility of CodeCover for various coverage needs of Java software. The source code instrumenting tool was used to instrument Cooja. The ant build.xml was modified appropriately to instrument the Cooja tool source code. Since the regression test suites were meant for Contiki and Cooja both, the coverage metrics of the Cooja tells the effectiveness of the existing test suite. The regression test suite was run as usual while CodeCover collected the data in the background. Since the regression test cases are executed in batch mode, the Java Virtual Machine

IJSER © 2019
http://www.ijser.org

(JVM) exits each time creating separate clf (coverage log file) for each of the test case. The Eclipse mode of testing was unsuitable for the existing test suites. Ant mode of instrumentation was apt as the Cooja already had the build.xml file. Since the tool created many clf files, the task at hand was getting the consolidated picture of the coverage. The command line utilities helped to achieve the intended task.

The following sequence of operation is carried out:

- Analyzing the clf files.
- Merging the sessions.
- Generating the report.

The generated report in hierarchical html format is as shown in the figure below.

It is observed that, the overall coverage was less than 20% for the base regression test suite.

7.5 Results

- The coverage in Cooja is less than 20% for the base test suite of Contiki.
- The ACTS tool generated test cases are more evenly distributed across motes and functionality wise.

7.6 Conclusion

IJSER

Each of the test cases generated as covering array in the output of the ACTs tool can be mapped to the actual test case of Contiki. This means writing the test cases in csc format for the Contiki operating system. Further the coverage data can be gathered using the CodeCover.

IJSER © 2019
http://www.ijser.org

8 Test Suite Design Methodology using Combinatorial Approach for Internet of Things Operating Systems

8.1 Introduction

In this chapter we describe how the test design can be done by using the Combinatorial Testing approach for internet of things operating systems. Contiki operating system is taken as a case study but we discuss what can be the approach for RIOT and Tiny OS operating systems. We discuss how the combinatorial coverage measurement can be gathered in addition to the traditional metrics code coverage. The test design generated by using Advanced Combinatorial Testing for Software is analyzed for Contiki operating system. We elaborate the code coverage gathering technique for Contiki simulator which happens to be in Java. We explain the usage of Combinatorial Coverage Measurement tool. Although we have explained the test design methodology for internet of things operating systems, the approach explained can be followed for other open source software.

Our previous chapter touches upon the test design using combinatorial testing approach for Contiki operating system [34]. In this chapter we intend to extend the concept and explain what can be done for the Internet of Things (IoT) operating systems which do not have standard regression test suites viz. RIOT and Tiny OS. We analyze the Advanced Combinatorial Testing for Software (ACTS) generated test suite design and explain how the traditional effective metrics, code coverage can be gathered in addition to more relevant combinatorial coverage measurements using combinatorial coverage measurement tool (CCM).

8.2 Typical Workflow for Baselining the regression Test Suite

Figure 11. depicts the typical flow of work when we want to ascertain the effectiveness of the test suite using combinatorial approach. First step is choosing the operating system for Internet of Things. Then traverse through

IJSER © 2019
http://www.ijser.org

the source code of the open source code base folders to see if the test suite exists. If it exists gather the coverage data using the code coverage tools.

If the gathered data indicates inadequate test suite, redesign the test suite using the combinatorial approach and gather the data. If the coverage data is less, it calls for re-visiting the test design. Base line the test suite once the adequate test criterion is met.

As can be seen from the diagram it can be iterative process. We document the process that was used for case study operating system in further sections. We describe the approach to be followed when the test suite already exists and when it does not. Section 3 is for the case when the base test suite already exists and Section 4 is for the case when the test suite does not exist.

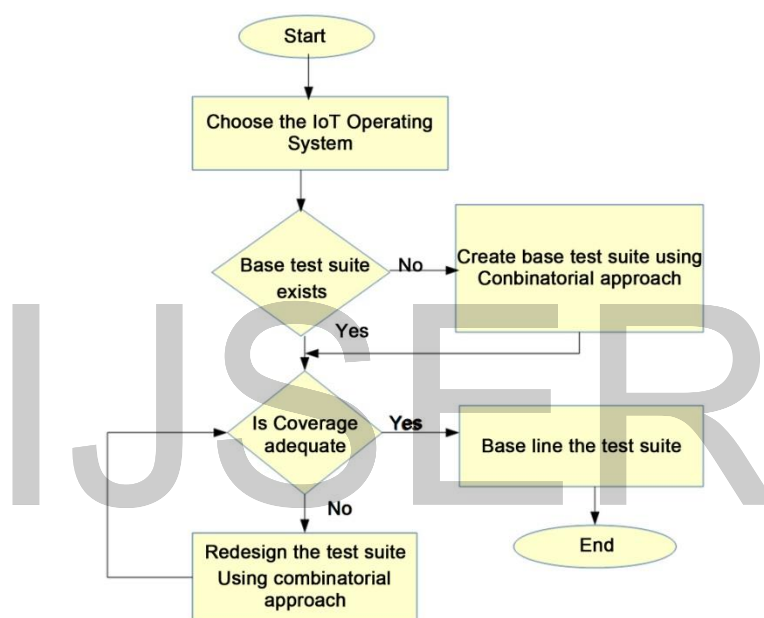


Figure 11. Typical work flow for base lining the test suite

8.3 Process of Redesigning the Regression Test Suite if it Already Exists

Figure 12 depicts the process in the case when test suite already exists. This section is for the case when the base lined test suite already exists as in the case of Contiki operating system version 2.7. We can use the either parameter based re-design or configuration based re-design as explained in the book and manual or combination of both. The coverage can be gathered using CCM and traditional coverage tools such as CodeCover. We did preliminary investigation

IJSER © 2019
http://www.ijser.org

using freely available tool CodeCover for the existing test suite. The coverage was less than 20%. Appendix B gives the data gathered using the CodeCover.

Then we visited the existing test suite to know the reason for low coverage. Few areas of improvements were observed in the existing test suite.

- No formal test design document existed.
- It appears that the test cases were concentrated around few mote types (hardware or configurations in the context of combinatorial testing).

We went through the whole regression test suite to extract the configurations supported and input parameters being used. We came up with Table 1 to be populated in the ACTS test model. When the ACTS was populated using these set of values, the generated test design document is as shown in Appendix A.

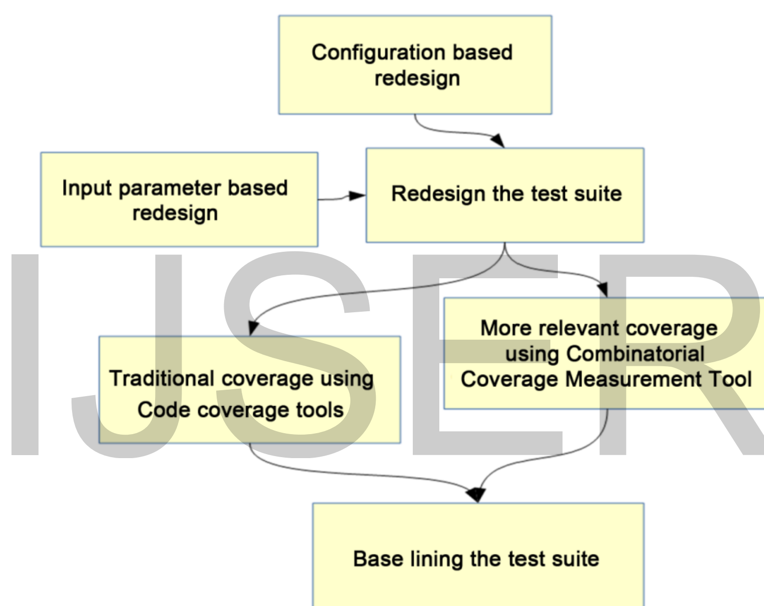


Figure 12. Process of base lining the test suite if it already exists

8.3.1 Contiki Specific Details

Contiki is open source operating system widely used and accepted for Internet of Things. It has base-lined regression test suite for version 2.7. Contiki gives the user friendly operating system in the form of instant Contiki which has Ubuntu like the feel with the tool chains to make the iterative development easy. The developers can use the instant Contiki to test the patches and testers can use the same environment for ascertaining the reliability of the operating system without procuring the hardware for all the mote types.

IJSER © 2019
http://www.ijser.org

Contiki gives the simulator which is called Cooja. The Cooja simulator talks to the Contiki using Java Native Interface (JNI). The test cases are called csc files which are understandable by Cooja. We found Eighty three test cases of this type in the regression folder. However, these test cases were concentrated around few mote types.

As already mentioned, Appendix A gives the test design generated using ACTS for Table 1 input. Let us visit the column 2 of the design. We can see that the generated test cases are spread across the mote (hardware) types. Further, the generated test design takes care of the input parameters as well for the test cases.

Now the task at hand is mapping these generated test cases to functional test cases (xml files called csc) which are understandable by Cooja and gathering the coverage data again. The coverage data should improve in principle. We are working on this.

8.4 Process of Designing the Regression Test Suite if it Does Not Exist

IJSER

Figure 13 depicts the case when test suite does not exist. This process is more suited for operating systems which do not have standard regression test suite viz. RIOT and TinyOS. Since the functional specification and test design are both missing in case of these operating systems, we will have to come up with the functional specification document first. This will be our understanding of the functionality that these operating systems support. Once the functionality of these operating systems is understood we will have to come up with the test design. Configuration to be supported and input parameters to be supplied for each test case will act as starting point for populating the ACTS test model. Once test design is generated, we will have to understand the test environment for these operating systems and the test design need to be mapped to functional test cases to be executed for gathering the coverage data. The CCM coverage will not be appropriate as the test cases generated using ACTS tool will always give 100% combinatorial coverage. Traditional coverage such as code coverage may be handy.

8.5 Contiki Specific Environment Changes to be Done

IJSER © 2019
http://www.ijser.org

In this section we document the changes that we did in the Contiki environment for the tasks at hand. Since we get implementation specific for case study operating system, this section can be conveniently skipped by the readers who are not interested in specific details for given operating system.

1. Log in as user in the instant Contiki environment.
2. Search for the .travis. yml
3. Add the build type you are interested in:

- BUILD_TYPE = “ipv6-apps”
- BUILD_TYPE = “CT”

-BUILD_TYPE = “compile-8051-ports”

4. Under the directory../contiki-2.7/regression-tests create a folder 02-CT
5. Under contiki-2.7/regression-tests/02-CT directory create *.csc files you are interested in viz.

01-custom.csc 02-custom.csc

6. The Make file should look like include../Makefile. simulation-test
7. Create a 01-custom.csc file in the Cooja tool. Use the test script editor to create a java script which will be essential while running the test case from command line using the makefile.

IJSER

IJSER © 2019
http://www.ijser.org

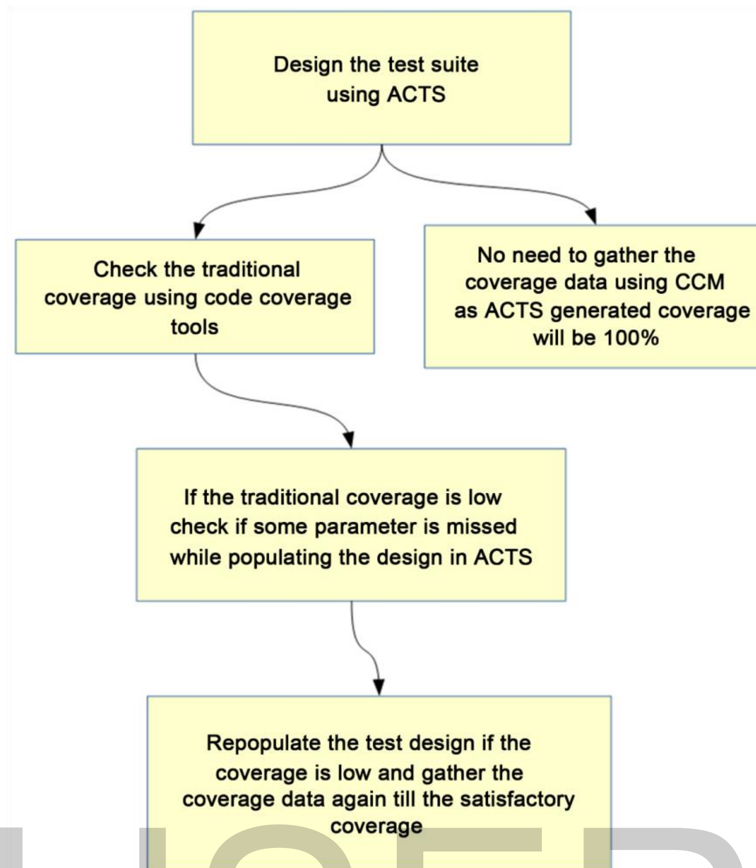


Figure 13 Process of baselining the test suite if it does not exist

IJSER

8. Modify the build.xml suitably as explained in Appendix C.
9. Run the regression test suite as usual.
10. Test run will create many *.clf files.
11. Create a script for analyze, merge and generate report.

8.6 Conclusion

In this chapter we presented the approaches that could be employed for designing the regression test suite using combinatorial approach. We explained how the bench marking of the regression test suite could be done using the traditional approaches such as code coverage in addition to coverage gathered using combinatorial coverage measurement tools.

IJSER © 2019
http://www.ijser.org

9. CT-RTS: Contiki and Cooja Regression Test Suites Design and Implementation using Combinatorial Testing

9.1 Introduction

This chapter we propose a mechanism of augmenting existing regression test suite by additional tests generated from a combinatorial approach. We then use our approach on to build a practical and reliable regression test suite for Contiki OS.

9.2 Background

IJSER

Contiki is a widely accepted Internet of Things operating system which is suited for memory and resource constrained devices. Contiki is open source software with a substantial user community. Contiki software comes with Instant Contiki which is an user friendly environment for testing. Using VMWare the Instant Contiki can be launched in a desktop environment. The Ubuntu based environment comes with tool chain dependencies which helps in making incremental changes to the operating system easy. Contiki can be built for various target platforms by tweaking the make file. Contiki supports several hardware platforms. The Instant Contiki has built in Java Simulator tool called Cooja which talks to the Contiki using the Java Native Interface (JNI).

Cooja has standard regression test suite in the regression test folder which are basically XML files with csc extension. These csc files are understandable by Cooja. The XML files have information of configuration and arrangement of mote type along with scenario specific java script embedded in them. Although Contiki supports the various hardware platforms, the regression test suite doesn’t reflect it. The test cases are concentrated around few mote types.

IJSER © 2019
http://www.ijser.org

When we did preliminary investigation, we noticed scope for improving the test suite. Two additional test suites were planned. These additional test suites are called reengineered test suite and Cooja test suite respectively.

Table 9. gives more details about the test suites.

Table 9 Test suites and their description

Test suite	System under test	Additional comments
Base test suite	Mainly Contiki	This can be found in Regression folder. This is referred as “Test suite A” in this chapter.
Re-engineered test suite	Mainly Contiki	This suite is created by picking additional test cases from ACTS. This test suite is referred as “Test suite B”.
Cooja test suite	Mainly Cooja simulator	This test suite is created from scratch using ACTS. This test suite is referred as “Test suite C”

IJSER

We wanted to quantify the effectiveness of test suites by gathering the coverage data in operating system and its simulator for the cases A, B and in simulator in case C. However, there are no open source or proprietary code coverage tools for software written in C language for all the target platforms supported by Contiki. Further, Contiki is a hard real time operating system. C code coverage tools add the overhead in the form of probes or traces making the test cases to fail. We explored J Test Pro [35] and G Cover [36]. They do not meet the requirement as the supported hardware platforms are different from the supported hardware platforms of Contiki or they make use of proprietary compilers. Therefore we decided to get the indirect measure of coverage in simulator for case test suites A and B. For test suite C since the system under test is simulator the coverage was gathered directly on the simulator written in Java.

9.2.1 Existing regression test suite

Unlike other IoT operating systems viz. RIOT [12] and Tiny OS [10], Contiki operating system comes with the standard regression test suite which we refer

IJSER © 2019
http://www.ijser.org

as base regression test suite. The base regression test suite containing 64 test cases are used for regression testing of Contiki. We did the initial analysis of the regression test suite and found that the test cases were not evenly distributed across hardware platforms supported.

The lack of testing across hardware platforms is serious issue because this is an OS on which many applications are planned. A lack of well tested code results in applications not working due to problems in the OS.

Further there is no requirement specification document for the Contiki Operating system and the test design document is unavailable. In such case, the existing regression test suite was starting point to understand the functionalities supported by the Operating system.

9.2.2 ACTS tool for generating combinatorial test design

In this work we use NIST ACTS tool for generating the test design. We applied the ACTS tool to generate the test cases for Cooja. Our preliminary analysis showed that the ACTS generated test cases were evenly distributed around hardware configurations. The input that needs to be supplied to the ACTs tool is given in Appendix A. These are the parameters and parameter values of Cooja along with constraints that needs to be supplied to ACTS tool. ACTS generates its output in a series of rows. Each row represents a test case. These test cases are then mapped to the functional test cases, these can be actually executed in the test environment.

9.2.3 Code coverage using OpenClover

Code coverage gives a quantitative measurement of how well the test cases are testing the software. Various Java coverage tools exist today. The coverage tools employ either source code or byte code instrumentation for gathering the coverage data. Instrumentation is a process where a tool inserts additional hooks into the codebase which it later uses for gathering the data. The coverage was meant for both the Contiki operating system and its simulator Cooja, here we show the coverage data on Cooja. For Cooja, build.xml already exists. Therefore, the ideal candidates for our study were CodeCover and Clover [24].

The regression test cases are written such that the Java Virtual Machine (JVM) comes up and terminates for each test case. The coverage tool CodeCover when used will generate the coverage log file per session of the

IJSER © 2019
http://www.ijser.org

IJSER

run. This means if there are hundred test cases, there will be hundred coverage log files (CLFs). To get the consolidated view at the regression test suite level these sessions have to be merged. The merging of hundreds of sessions is ineffective in CodeCover. We find that Clover can do the merging in an effective manner. Clover is open source software effective April 2017. Clover augments the database file for various sessions automatically. This means there is no need to merge the sessions either manually or through the shell scripts. We used the Clover Java code coverage tool for gathering the coverage data. The build.xml was modified appropriately for the activity of code coverage gathering.

9.3 Re-engineering the base test suite

For the software under study two scenarios exist.

- The software has the standard regression test suite with it.
- The base regression test suite missing.

For the first case the missing test cases can be generated using the CCM tool. The other approach is to design the test suite using the ACTs and augment the missing test cases to the regression test suite by finding the missing test cases from the regression test suite. In the second case the design will be using ACTs and choosing the appropriate number of test cases as per the coverage needs.

IJSER

In this section we show how the inadequacy of the coverage data can be addressed by re-engineering the regression test suite.

We generated the test design using the ACTs tool of NIST. The ACTs tool distributed the test cases evenly around the mote types. The test cases of Contiki and Cooja are in *.csc format which are basically xml files understandable by Cooja. The *.csc files are scenario specific and typically few hundreds of lines in length. In our earlier work we generated the design using ACTs tool the test design suggested even distribution of test cases around micaz, esb, wizmote and z1 mote types in addition to sky and contikimotetype.

Figure 14. depicts the idea of gathering the bench mark code coverage data. The code coverage data of the baselined regression test suite is compared with the re-engineered test suite code coverage data.

IJSER © 2019
http://www.ijser.org

We have described the ACTs design with all the parameters taken at a time [18]. The implementation of such test cases is impractical and therefore two parameters were taken at a time. Appropriate constraints were introduced in the ACTs design to make it possible.

The test design was implemented using the NIST ACTs tool. The input parameters were decided after going through the base regression test suite. The ACTs tool suggested two hundred and eighty nine test cases.

The functional test cases build the firmware from the *.c files in examples directory and copy them in motes for a given scenario.

For example:

1) example-runicast.c in directory /home/user/contiki-2.7/examples/rime is successfully building for all target types viz. sky, esb, exp5438, z1, wismote and micaz.
2) The same behavior as in 1) is expected from example-trickle.c in directory /home/user/contiki-2.7/ examples/rime. However, the build that is 'make command' is successful for sky,esb,z1 and wismote but make command is failing for exp5438 and micaz.

The behavior in 1) and 2) is external to the test cases that we are implementing. Meaning the test cases depend upon the successful build for all the target types.

What this means is we will not be able to implement all the test cases from ACTs design. Further, we wanted to restrict the number of test cases to reasonable count say 100. This is for a reason. If all the 289 suggested test cases of ACTs are implemented and included in the regression it would mean

- Test all approach
- Very long execution cycles given the time taken to execute the test cases in Contiki and Cooja environment.

IJSER

IJSER © 2019
http://www.ijser.org

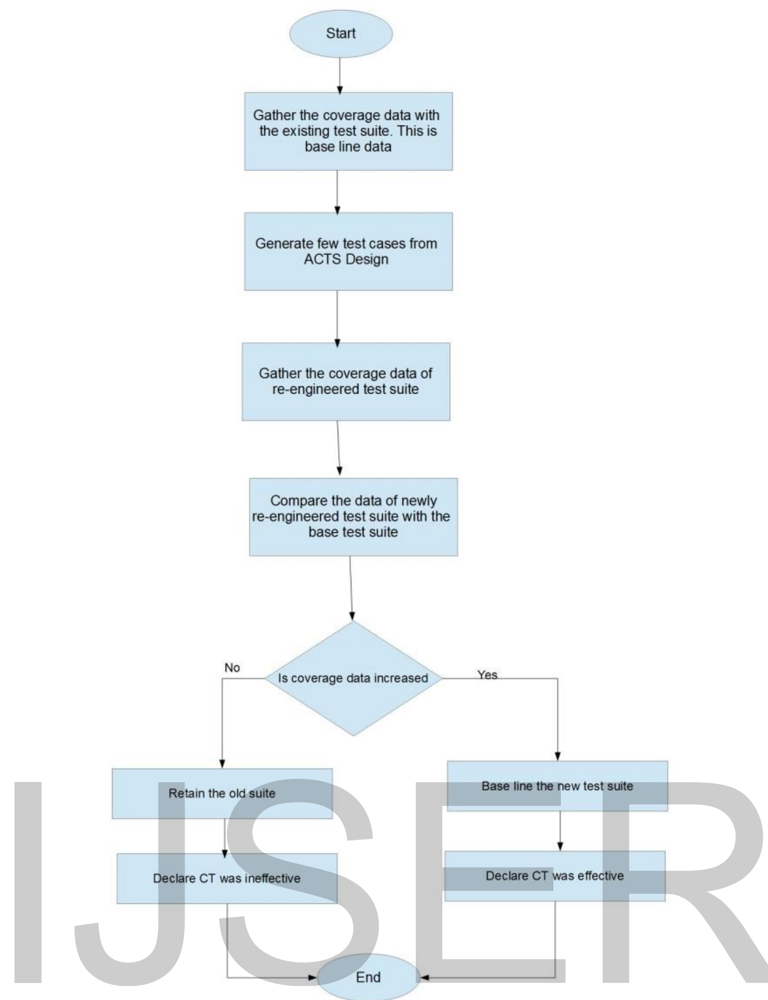


Figure 14 Process of gathering code coverage for CT

9.4 Test design using ACTS tool for re-engineered test suite

The ACTs suggested test cases were super set of the base regression test suite test cases. Since the base regression test suite already had sixty four test cases, additional thirty five test cases from the ACTs design were implemented using the auto generation. Essentially we have two test suites:

- Base regression test suite that comes with Contiki which has 64 test cases.
- Modified regression test suite with ninety nine test cases taken from ACTs design. Out of ninety nine test cases, thirty five test cases are new and remaining 64 test cases are

IJSER © 2019
http://www.ijser.org

same as that of base regression test suite.

9.5 Auto generation of test cases

As mentioned in the section three, thirty five additional test cases were introduced to the base regression test suite. Since the functional test cases exceed hundred lines of xml code, additional thirty five test cases translate into more than three thousand lines of xml code. Since the effort was substantial, auto generation of test cases was explored. The idea is to take the human readable text file and auto generate the functional test cases. The functional test cases have mote arrangement specific information along with scenario specific java script. The mote and mote arrangement information was auto generated using the tool that we developed. Scenario specific java scripts were introduced manually.

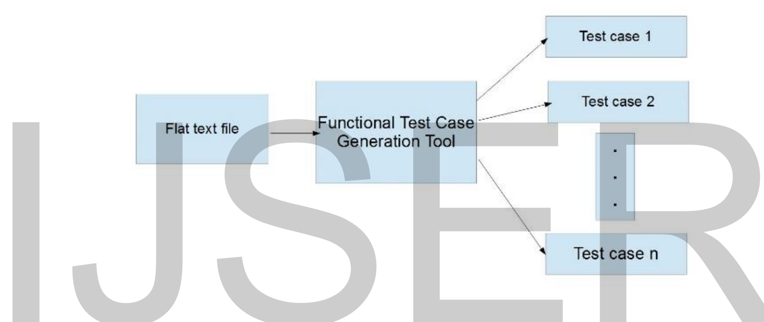


Figure 15. Functional test case auto generation tool

Figure 16 below shows the process used for auto generation of the test cases. Each stage output acts like input to the stage next to it. To begin with we developed tool which is eight seventy eight lines of code. The code was written in Java and reuses the Cooja code at several places for the generic engine. The generic engine was coded first. The engine needed drivers to drive it. The RegEx package is used to parse the input text file. The parsed information is used to populate the internal data structures of the tool. The driver then uses this data to drive the generic engine. In summary, the input text file contains all the configuration information of mote types readily embedded in it. The output files will be csc XML files which will have the configuration information needed for the test cases. The scenario specific java script is then embedded manually in the test case to complete it. The XML line

IJSER © 2019
http://www.ijser.org

count for the activity was in excess of three thousand and five hundred for the activity.

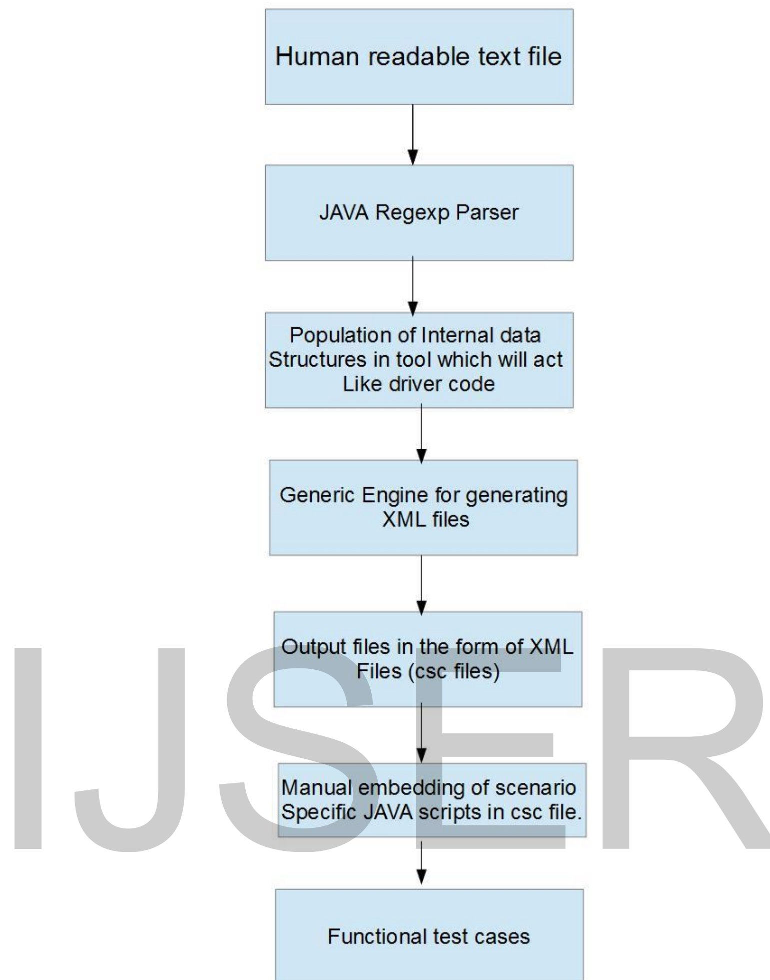


Figure 16. Test case auto generation process

IJSER

```
(
{ testcasename,test1.csc }
{ title,firsttestcase}
{ radiomedium,se.sics.cooja.radiomediums.UDGM}
{ motetype,se.sics.cooja.mspmote.SkyMoteType,source,[CONTIKI_DIR]/examples/netperf/netperf-shell.c,identifier,sky1}
{ mote1,se.sics.cooja.mspmote.SkyMote,motetype_identifier,sky1}
{ SimControl,placeholder}
{ TimeLine,placeholder }
)
(
{ testcasename,test2.csc }
{ title,secondtestcase}
{ radiomedium,se.sics.cooja.radiomediums.UDGM}
{ motetype,se.sics.cooja.mspmote.SkyMoteType,source,[CONTIKI_DIR]/examples/netperf/netperf-shell.c,identifier,sky1}
{ mote1,se.sics.cooja.mspmote.SkyMote,motetype_identifier,sky1}
{ SimControl,placeholder}
{ TimeLine,placeholder }
)
.
.
.
(
{ testcasename,testn.csc }
{ title,nthtestcase}
{ radiomedium,se.sics.cooja.radiomediums.UDGM}
{ motetype,se.sics.cooja.mspmote.SkyMoteType,source,[CONTIKI_DIR]/examples/netperf/netperf-shell.c,identifier,sky1}
{ mote1,se.sics.cooja.mspmote.SkyMote,motetype_identifier,sky1}
{ SimControl,placeholder}
{ TimeLine,placeholder }
)
```

IJSER

Figure 17. Sample input text file for the tool

Figure 17. shows the sample input text file which will be accepted by the tool. The parser is written such that any number of cscs could be generated in one run. The logical blocks for the individual test cases are called records and each line within the record is field. The engine is called generic since the engine works for any number of mote types and motes. The code written for this work can be found in git hub [37].

Figure 18 below shows the sample output xml file generated using the tool. As can be seen from the generated xml the xml is complete in all aspects except for the scenario specific java script. Since the generation of scenario specific information cannot be automated, the process of inserting the java script is manual.

In our case the test case generation was a two pass mechanism. The skeletons of the XMLs were generated in first pass and the java script was inserted in second pass resulting in the complete test cases ready to be executed in Cooja environment. The advantages of automation is basically eliminating the manual work and elimination of human errors while coding the individual lines of XML.

IJSER © 2019
http://www.ijser.org

```
<?xml version="1.0" encoding="UTF-8"?>^M
<simconf>^M
 <simulation>^M
  <title>firsttestcase</title>^M
  <randomseed>123456</randomseed>^M
  <motedelay_us>1000000</motedelay_us>^M
  <radiomedium>^M
   se.sics.cooja.radiomediums.UDGM^M
   <transmitting_range>50.0</transmitting_range>^M
   <interference_range>100.0</interference_range>^M
   <success_ratio_tx>1.0</success_ratio_tx>^M
   <success_ratio_rx>1.0</success_ratio_rx>^M
  </radiomedium>^M
  <motetype>^M
   se.sics.cooja.mspmote.SkyMoteType^M
   <identifier>sky1</identifier>^M
   <description />^M
   <source EXPORT="discard">[CONTIKI_DIR]/examples/netperf/netperf-shell.c</source>^M
   <commands EXPORT="discard" />^M
   <firmware EXPORT="copy">[CONTIKI_DIR]/examples/netperf/netperf-shell.sky</firmware>^M
   <moteinterface>se.sics.cooja.interfaces.Position</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.RimeAddress</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.IPAddress</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.Mote2MoteRelations</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.MoteAttributes</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspClock</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspMoteID</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspMoteID</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyButton</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyFlash</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyCoffeeFilesystem</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.Msp802154Radio</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspSerial</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyLED</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspDebugOutput</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyTemperature</moteinterface>^M
  </motetype>^M
  <mote>^M
   <breakpoints />^M
   <interface_config>^M
    se.sics.cooja.interfaces.Position^M
    <x>0.0</x>^M
    <y>0.0</y>^M
    <z>0.0</z>^M
   </interface_config>^M
   <interface_config>^M
    se.sics.cooja.mspmote.interfaces.MspMoteID^M
    <id>-1</id>^M
   </interface_config>^M
   <motetype_identifier>sky1</motetype_identifier>^M
  </mote>^M
 </simulation>^M
</simconf>^M
^M
```


IJSER

Figure 18. Sample output XML file

IJSER © 2019
http://www.ijser.org

9.6 Test design for Cooja test suite using ACTS tool

Since the test suite was to be created from scratch, the functionality of Cooja was studied to begin with. This was the starting point. To keep the test cases count reasonable, the input parameter modelling was done in such a way that the main model was broken down into 4 sub models. The models are documented in Appendix B for reference. These input parameters were then supplied as inputs for ACTs to generate the test cases.

9.7 Code coverage data gathering process

Figure 14 depicts the process flow for gathering the coverage data prior and post CT. It depicts the comparison to be done. The coverage data of base regression suite is the reference point. The Figure 19 shows how the Clover interacts with the Cooja to generate the output files for inference. Code Coverage gathering process shows the changes to be done to the test environment for gathering the required code coverage data. The build.xml needs to be modified appropriately to incorporate the Clover in Cooja environment.

IJSER

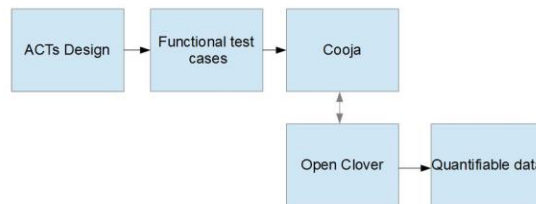


Figure 19 Cooja and Open Clover interaction

IJSER © 2019
http://www.ijser.org

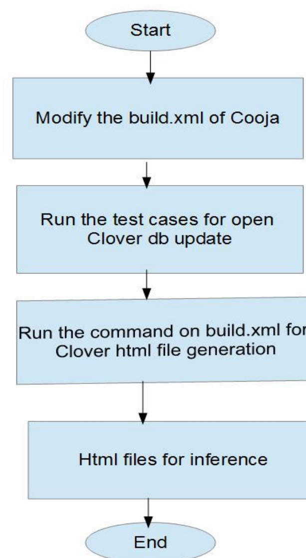


Figure 20. Test environment change for Clover data gathering

9.8 Results

Table 10, Table 11 and Table 12 gives the comparison of code coverage in the simulator for various java packages. TPC is the total percentage of coverage at the simulator level. This is as per the internal calculation of the coverage tool employed for gathering the coverage. Section 9 elaborates on how the TPC is calculated in the clover.

IJSER © 2019
http://www.ijser.org

Table 10. Code coverage in simulator for test suite A

Java package	Test suite A
Cooja.plugins.analyzers	0%
Cooja.plugins.skin	0%
Cooja.positioners	0%
contikimote.interfaces	3%
cooja.util	6.50%
cooja.motes	6.60%
cooja.plugins	6.60%
cooja.dialogs	7.30%
cooja.contikimotes	17.90%
cooja.interfaces	20.80%
se.sics.cooja	33%
cooja.radiomediums	43.60%
cooja.emulatedmotes	1.70%
TPC	13.6%

IJSER

Table 11. Code coverage in simulator packages for Test Suite B

Java package	Test suite B
Cooja.plugins.analyzers	0%
Cooja.plugins.skin	0%
Cooja.positioners	0%
contikimote.interfaces	3%
cooja.util	6.50%
cooja.motes	6.60%
cooja.plugins	6.90%
cooja.dialogs	7.70%
cooja.contikimotes	19.70%
cooja.interfaces	20.80%
se.sics.cooja	34.60%
cooja.radiomediums	44%
cooja.emulatedmotes	53.80%

IJSER © 2019
http://www.ijser.org

TPC	14.70%

Table 12. Code Coverage in simulator package for Test suite C.

Java package	Test suite C
Cooja.plugins.analyzers	77.80%
Cooja.plugins.skin	72.50%
Cooja.positioners	87.40%
contikimote.interfaces	52%
cooja.util	53.70%
cooja.motes	58.60%
cooja.plugins	72.50%
cooja.dialogs	69.80%
cooja.contikimotes	64.20%
cooja.interfaces	69.30%
se.sics.cooja	77%
cooja.radiomediums	60.70%
cooja.emulatedmotes	54.70%
TPC	70.50%

IJSER

Figure 22. shows that the coverage at the package level alone will not suffice. We would be further interested in knowing whether the testing at class level is adequate and how many classes have given percentage of coverage. Figure 9. is to aid such analysis. An ideal test output would have all the classes represented in the extreme right bar of the chart. The three bar charts are for test suite A, B and C respectively. As can be seen from the charts for the test suite C, the bars on the right are taller. Which indicates the testing in simulator was more adequate in test suite C than in A and B.

Figure 23. is tree map for the test suites A, B and C. The convention used for the tree maps is as follows:

- Deep red no coverage.
- Pale green full coverage
- Yellow lies between red and green

IJSER © 2019
http://www.ijser.org

- Square size indicates the complexity of the code.

The coverage treemap report allows simultaneous comparison of classes and package bycomplexity and by code coverage. This is useful for spotting untested clusters of code.The treemap is divided by package (labelled) andthen further divided by class (unlabelled). The size of the package or classindicates its complexity (larger squares indicate great complexity, while smallersquares indicate less complexity).

Colours indicate the level of coverage, as follows:

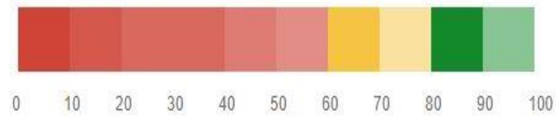
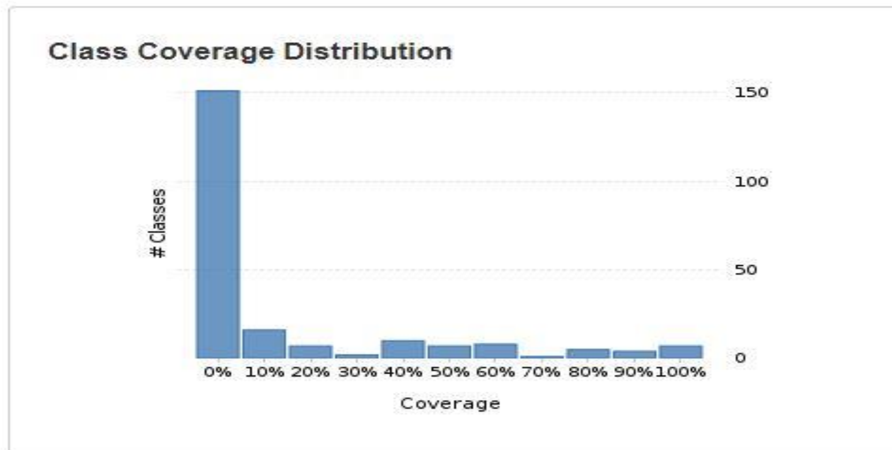
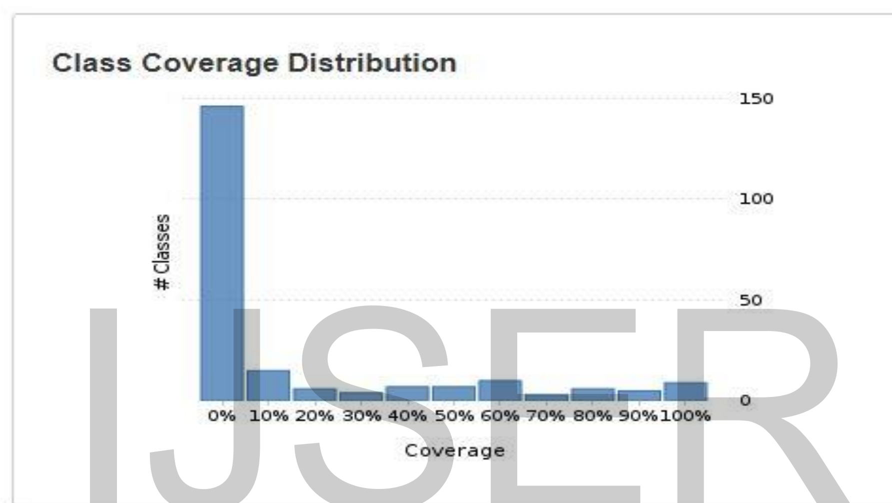


Figure 21 Reading the treemap

IJSER

IJSER © 2019
http://www.ijser.org





IJSER

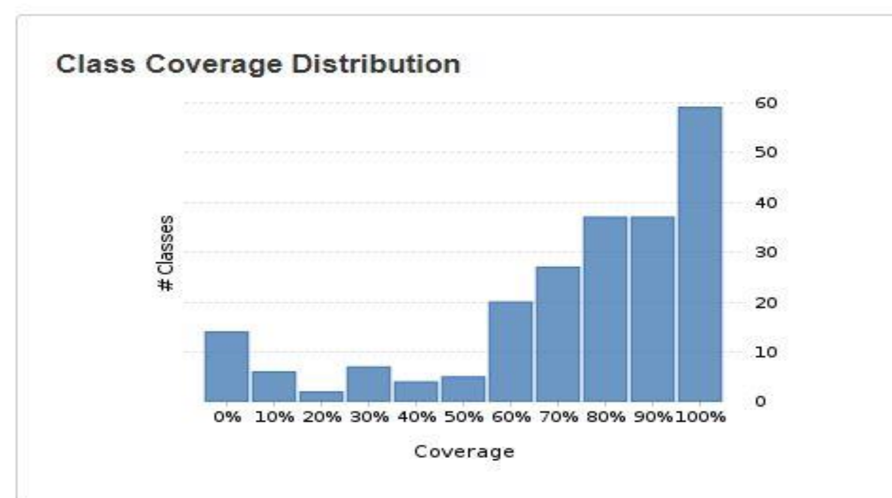


Figure 22 Class coverage distribution in simulator for three suites A, B and C respectively.

IJSER © 2019
http://www.ijser.org

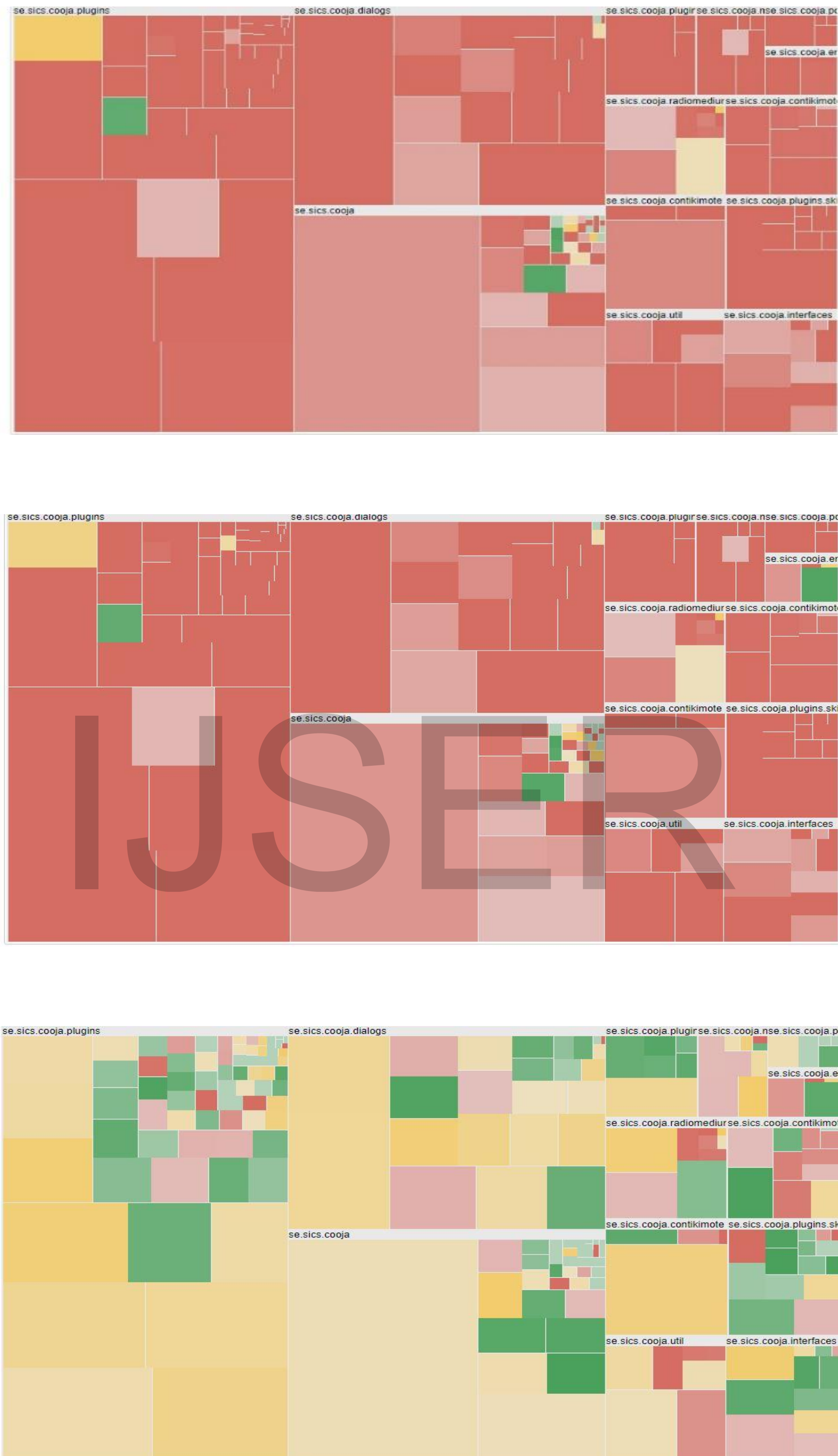


Figure 23. Tree maps of code coverage in simulator for three suites A, B and C respectively

IJSER © 2019
http://www.ijser.org

9.9 Results analysis

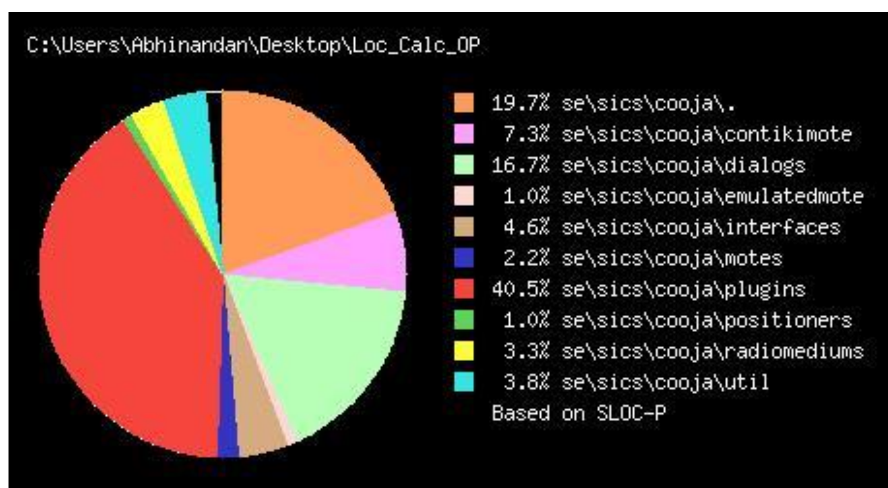


Figure 24. Source code analysis of simulator with LOCMetrics

Figure 25. Code metrics of Cooja code base

We saw Total Percentage of Coverage (TPC) of around 13.6% for test suite A and 14.7% for test suite B. This low percentage of coverage needs to be explained to begin with. We analyzed the code of Cooja and noticed that the major packages are plugins, cooja package and dialogs. Cooja supports two modes of execution viz. Graphical User Interface (GUI) mode and non GUI mode. The Regression test suite environment is written such that the Cooja runs in non GUI mode. The packages plugins, cooja package and dialogs have significant code meant for GUI. In such cases achieving high TPC is not possible with test suites A and B.

Further CT in test suite B is meant to test various configuration combinations. The Cooja code is written such that the firmware file is built externally (The *.C files in examples directory are compiled) and loaded into

IJSER © 2019
http://www.ijser.org

mote types. This logic is mainly in the files se.sics.cooja.Contikimote.ContikiMoteType and se.sics.cooja.Simulation. Therefore inspite of adding the CT test cases to test various hardware configurations we didn't see the increase in code coverage of simulator.

We basically have three test suites:

1) Base test suite with 64 test cases. This is test suite A.
2) Re engineered test suite with base 64 + 35 ACTs test cases. This is test suite B.
3) Cooja test suite designed from scratch. This is test suite C.

We saw a marginal jump of 1.1% in TPC between test suite A and test suite B. Where TPC is calculated as follows in Clover:

TPC = (BT + BF + SC + MC)/(2*B + S + M) * 100%

where

BT - branches that evaluated to "true" at least once
BF - branches that evaluated to "false" at least once
SC - statements covered
MC - methods entered

B - total number of branches
S - total number of statements
M - total number of methods

As we have the B, S and M numbers from the clover output, total jump in (BT+BF+SC+MC) is 312 for 35 added test cases.

Further, each package responded in different way for the 35 added test cases. For example, the emulated motes package registered a jump of 52.1% for additional 35 test cases of test suite of B.

IJSER © 2019
http://www.ijser.org

The low TPC is attributed to the way the Cooja runs in the regression mode. In regression, Cooja runs in non GUI mode and significant chunk of the code in Cooja is for GUI.

Further, the code of the Cooja is such that, for various emulated configurations of CT, the firmware is built externally and loaded into mote types. Rest of the code is common for various mote types. With such type of tool, the TPC increase will be low but that will not be a direct measure of effectiveness of CT of test suite B.

For test suite C, system under test is the simulator i.e Cooja. In this mode the intention is to test the simulator thoroughly. The test suite C runs the simulator in both modes GUI and non GUI. The test cases of ACTs output are executed manually. For test suite C, we mainly concentrated on success path test cases. Few critical failure path test cases were executed. A quick look at the Cooja code reveals 237 catch blocks of Java code. These correspond to 237 failure scenarios. We did not hit all the failure paths. We decided to conclude the refining of input parameter modelling at 70% coverage.

IJSER

9.10 Supplementary material

Clover output of the test executions are kept in the repository and can be accessed online [32].

9.11 Conclusion

The base regression test suite of Contiki and Cooja was redesigned using the combinatorial approach. Auto generation of functional test cases was explored and test cases were generated and added to the base regression test suite. We found that the increase in coverage of the simulator was marginal for the reengineered test suite because of the execution mode of simulator and simulator code structure. However, for the test suite which was specifically designed using CT with simulator as the system under test showed substantial increase in the simulator coverage.

IJSER © 2019
http://www.ijser.org

10. Combinatorial Testing based Functional Test Case Generator for Contiki Operating System and Cooja Simulator

10.1 Introduction

Evolving multi-parameter, multi-configuration systems require regression test suite that can be customized. This is in terms of run time. Run time can be customized by generating the combinations using combinatorial techniques. For systems like Contiki operating system, the test cases need to be executed in its simulator Cooja. Executing test cases in a simulator requires functional test cases to be generated from the combinatorial parameter combinations obtained. In this work we present a methodology to generate the functional test cases. We present Functional Test Case Generator for Contiki and Cooja (FTCGCC), which is a tool developed using our methodology. We demonstrate use of our tool by generating customizable regression test suite for Contiki and Cooja using code coverage as criteria. FTCGCC is developed for the test case generation when target System Under Test is IoT operating system Contiki and its simulator Cooja. We find that our tool generates all the test cases. FTCGCC generates the cases which are readily executable in the Contiki and Cooja environment. Further, the approach mentioned can be used in other cases where the simulators are involved which accept the XML based test cases. The design of the FTCGCC can be reused for other simulators. The FTCGCC test case generator engine is generic in nature.Combinatorial testing (CT) is field of testing which is in practice both in the industry and research [1]. When the System Under Test (SUT) has combination of configurations or input parameters, CT can be used to reduce the number of regression test cases needed [2] [3]. National Institute of Standards and Technology (NIST) gives the tools which aid in performing combinatorial testing. Advanced Combinatorial Testing for Software (ACTS) is one such tool [4]. ACTS tooltakes the input parameter and input parameter modeling and generates the test design

IJSER © 2019
http://www.ijser.org

IJSER

document which is in the form of rows. Each row is independent test case. For the cases when the test cases are readily executable as in the case of Graphical User Interface (GUI) applications, no intermediate step is required. However, when the test environment expects the test cases in a particular format, intermediate processing is required. For the System Under Test (SUT) as Contiki operating system and its simulator Cooja, the test design contains the test cases which are not readily executable in the SUT test environment. Manual generation of functional test cases is a tedious and error prone task. In this work we present our tool, Functional Test Case Generator which can be used for auto-generation of functional test cases. We demonstrate the successful application of our tool to generate the functional test cases for the Contiki operating system. The novelty of the FTCGCC is that it is a tool developed for the specific requirements of the test case generation for Contiki and Cooja. We evaluated few generic purpose test case generators such as IBM ATG, TCGTool, Randoop, Automatic Testing Platform, Conformiq. These are generic purpose tools. These tools generate the test cases either on the basis of code or requirement. However, they do not meet the requirements of functional test case generation for Contiki and Cooja. We therefore had to develop the FTCGCC from scratch for this work. The FTCGCC takes the text file as input and generates the test cases which are readily executable in Contiki and Cooja based test environment as explained in the subsequent sections of this work. In this work we give the details of the high level design and software implementation of the tool in further sections. The usage of the tool and final results of the test case execution are also documented.

Table 13. Various existing test case autogeneration tool

Test case generator	Owner	Remarks
IBM ATG	IBM	Model Based Testing tool
TCGTool	SourceForge	Test Case generation from finite state machines.
Randoop	University of Washinton	Junit test case generator
Automatic Testing Platform	SourceForge	Useful for the web applications on the client side
Conformiq	Conformiq	Test case generation from graphical model.
Blueprint	Blueprintsys	Test case generation from requirements.

IJSER © 2019
http://www.ijser.org

10.2 Combinatorial testing and NIST ACTS tool

For real world software, the number of input parameters and their combinations can be very large making it impractical to develop test cases covering all the combinations of the input parameters. Combinatorial testing addresses this issue partially [2]. NIST offers ACTS covering array generator produces compact arrays covering 2-way to 6-way combinations.

We use the ACTS tool to generate the required combinations to generate a regression test suite. We use as SUT the Contiki operating system and its Cooja simulator. In Section III and IV we explain the details of the SUT chosen and the challenges in developing a practical regression test suite. Subsequent sections explain more about the ACTS tool usage and the details of our proposed tool for auto generation of the test cases from the ACTS output.

10.3 Contiki the IoT operating system

Internet of Things (IoT) comprises of things or devices with unique identities that are connected to the internet. The choice of the operating system for the device depends on the purpose of the node. A typical IoT network comprises of expensive nodes and inexpensive nodes. Inexpensive nodes just collect and forward the data to the nearest expensive node. Expensive nodes on the other hand do have few analytics capabilities in addition to the functionalities possessed by the inexpensive nodes. A generic IoT device supports Connectivity, Processor, Audio/Video interfaces, I/O interfaces for sensors and actuators, Memory interfaces and Storage interfaces.

Table 14. Node operating system

Node type	Example operating system
Inexpensive node	Contiki, RIOT and Tiny OS

IJSER © 2019
http://www.ijser.org

Expensive node	Arch Linux, Amazon FreeRTOS, Android Things, Rasbian Linux etc

The Operating system will have all these constraints into consideration while being developed. In addition the IoT operating system needs to support the protocols desired. The IoT protocol stack contains mainly four layers:

i. Link layer
ii. Network layer
iii. Transport layer
iv. Application layer

Table 15. IoT layers and protocols

Layer	Protocol examples
Link layer	Ethernet, Zig bee, Wi-Fi, Wi-max or long range communication protocol such as 3G/LTE/5G etc.
Network layer	IPV4, IPV6, 6 LOW PAN
Transport layer	TCP or UDP
Application layer	HTTP, COAP, Websockets, MQTT, XMPP or AMQP

IJSER

Contiki, as the device layer OS, supports the protocols mentioned above. Contiki has the functionality implementation for all the above mentioned protocols. To test the protocol functionality a regression test suite needs to contain the corresponding test cases. Contiki operating system needs be tested to ensure that all the claimed platforms are supported by the Contiki operating system. Contiki supports platforms such as Exp5438, z1, wismote, micaz, sky, sentilla-usb and esb among others. In addition, there needs to be test cases to for the other functionality such as file system support etc.

IJSER © 2019
http://www.ijser.org

10.4 Cooja simulator

The Cooja simulator in its current form was designed and developed by Fredrik Osterlind. Since then it has evolved by many contributors. The acronym Cooja is derived from Contiki Operating System Java Simulator. Originally the Contiki was developed for resource constrained Wireless Sensor Network (WSN) nodes [6]. The main goal of the Cooja simulator is extensibility. Cooja achieves it using the interfaces and plugins.

Interfaces is for node property such as

- Position of the node
- Button
- Radio transmitter

On the other hand plugin is used for interacting with simulation such as

- To control simulation speed
- To watch all network traffic between the simulated nodes

Since Cooja simulator supports several different simulation environments at a same time simulation of HetNets (Heterogeneous Networks) is possible. The advantage of using the Java is that the Java supports JNI (Java Native Interface) using which the simulator can talk to real Contiki operating systems.

The test cases of the Contiki/Cooja are in the form of XML files. These files have *.csc extensions. The test cases typically compile the firmware in the *.c format and embed the firm ware in the Simulator. The firmware compiled depends on the target platform. In this work we focus on the XML files with *.csc extension and their auto-generation for the task at hand.

10.5 Regression test suite of Contiki Operating System

We explored the publically available regression test suite of the Internet of Things (IoT) operating system Contiki [7] and noticed various combination of configuration and input parameters. Contiki supports various hardware configuration like mote types, its communication devices etc. [33]. Further, we noticed that the test cases were concentrated around few hardware (mote) types in the test suite [34].

Initial design of the test suite using ACTS distributed the test cases evenly around mote types. We needed the step which would convert the test cases

IJSER © 2019
http://www.ijser.org

output by the ACTS to functional test cases. The base regression test suite of Contiki and Cooja consisted of 64 test cases. We wanted to add the additional test cases to the base regression test suite to see how the code coverage varies with the additional test cases with code coverage tools [24]. The design was done in the ACTS tool. ACTS tool generated 289 test cases. We noticed that the 289 test cases were super-set of the base regression test suite. Since the execution time needs to be over-night, we decided to limit the test cases to 99. Additional 35 test cases need to be added.
There exist two types of scenarios as depicted in the Figure 7. when ACTS tool is involved.

- When the test cases are readily executable on the target SUT as in case of GUI SUT
- When the test cases in the ACTS design need to be converted into executable test cases called functional test cases using the intermediate step.

The former case is explored in detail and is documented in the chapter 7. The latter is investigated in this paper.
Since each test case requires more than 100 lines of XML code, for the additional 35 test cases, straight calculation gives more than 3500 lines of XML code. In addition, manually coding further means that the process could be error prone. Automation of the functional test case generation has resulted in development of Functional Test Case Generator for Contiki and Cooja (FTCGCC). The idea is the tool should take as an input the human readable text file and should generate the XML files understandable by the Cooja simulator. Figure 15. gives the functionality at a high-level.

10.6 Requirements for FTCGCC

High level requirements for the tool are as follows:

1. The FTCGCC should take the text file and generate the XML files which are in the *.csc format understandable by Cooja simulator
2. The text file consists of records separated by special characters “(“ and “)”
3. Each record maps to individual test case.

IJSER © 2019
http://www.ijser.org

4. Each line in the record separated by special character “{“ and “}” called field within the test case, should map to a logical block of the test case.
5. The input text file could contain any number of records and any number of fields within the records. The parser should be generic enough to handle this.

10.7 High level design of FTCGCC

The tool is designed such that it takes the input text file consisting of records and fields. The field and field values are separated by special character “,”. If the input file consists of N records, the FTCGCC outputs N test cases. The test case creation is two pass mechanism:

- In the first pass, complete test case except the JavaScript is created using the FTCGCC.
- Since the JavaScript is scenario specific, it should be coded manually. In the second pass, the JavaScript is embedded manually in the test case created in the first pass.

Figure 16. gives the blocks involved in the design of the FTCGCC. First the input file is split using the Java’s regexp package. This step splits the input file in the form of records and fields. Each record maps to one test case and each field within the record maps to logical block within the test case. Next, the data structures within the tool are populated as per the parsed input file. These data structures act as driver data. The driver drives the generic engine which is meant for generating the XML files. The Cooja code is reused heavily in the generic engine.

As can be seen from the block diagram, the output of one block drives the next. Or in other words, the output of one stage is input to the next stage.

Table 16. Stages and functionality of FTCGCC

Stage	Description
Stage 1	Creation of input file to FTCGCC
Stage 2	FTCGCC creates records and fields using the Java regexp package
Stage 3	FTCGCC populates the vital data structures which act as driver data

IJSER © 2019
http://www.ijser.org

IJSER

Stage 4	FTCGCC drives the generic engine using the driver data of stage 3.
Stage 5	FTCGCC outputs the skeleton XML files in *.csc format.
Stage 6	The scenario specific JavaScript is embedded in the XML
Stage 7	Fully functional test cases are ready

10.8 Software implementation

To design a tool with the requirements mentioned in the requirement section, we had to analyze all the existing *.csc files of the Contiki and Cooja. We came up with the generic *.csc template as depicted in Figure 27. We noticed the vital entities that drive the simulator of the Contiki. We came up with the template input file to be supplied to the tool and the same is depicted in Figure 18.

IJSER

IJSER © 2019
http://www.ijser.org

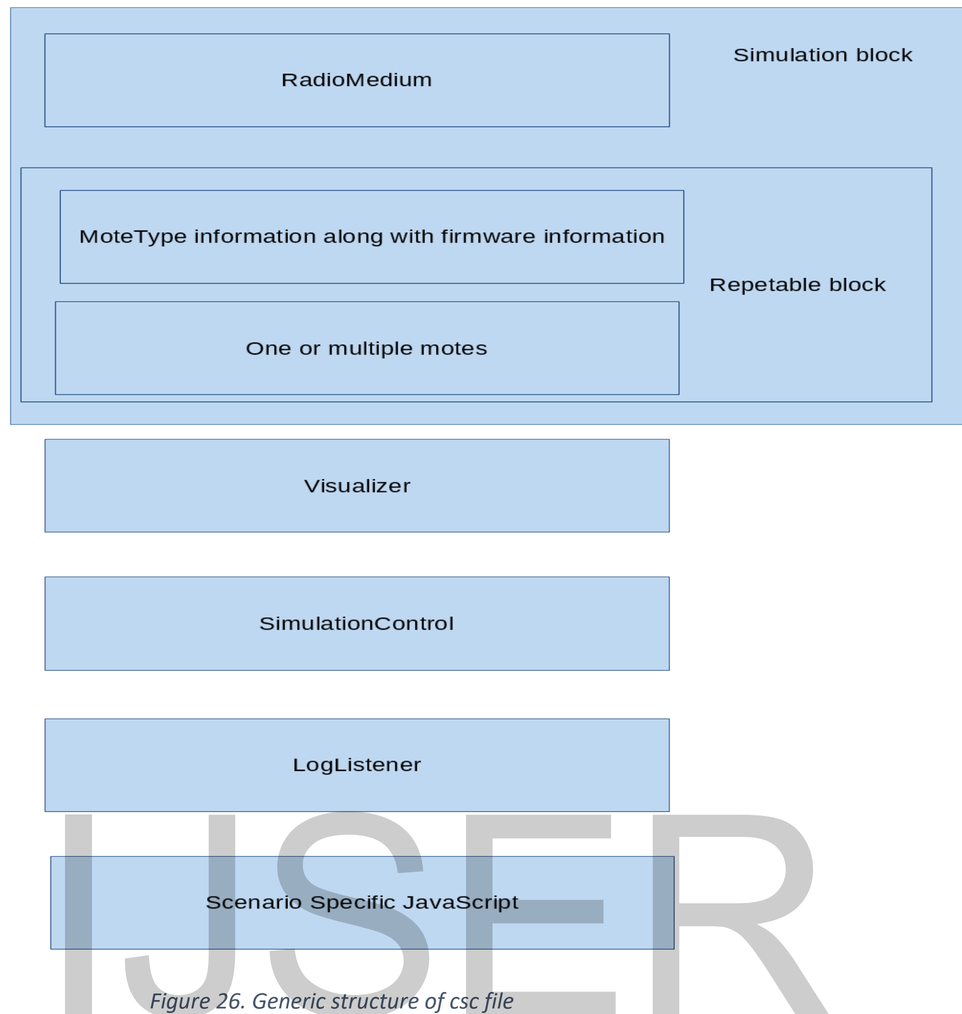


Figure 26. Generic structure of csc file

The csc which stands for the Cooja simulation configuration file contains the simulation configuration information understandable by the Cooja simulator. The simulation configuration mainly contains blocks that can be classified in the following categories

i) Simulation
ii) Plugins
iii) Scenario specific JavaScript

Simulation block contains radio medium information with mote type (along with firmware information) and mote information. Mote type and mote information is repeatable block.

IJSER © 2019
http://www.ijser.org

10.8.1 Java's regexp parser

The java.util.regex package deals with the sophisticated pattern matching operations. Regular expression is a string of characters describing character sequences. Regular expression also called pattern can be set of characters, wildcard character combinations along with various quantifiers.

Regexp is a powerful package for text processing. Particularly, following operations can be done very easily with the help of Regexp

- Text processing
- Text manipulation
- Tokenisation

The input file was designed such that:

- Each record which maps to test case starts with special character "(" and ends with ")"
- Each field which maps to logical block in the test case starts with special character "{" and ends with "}"
- Field name and field values are separated by special character ","
- We created the patterns for each of the unique character sequences mentioned above and we split the character sequences accordingly.

10.8.2 Java Document Object Model Parser

IJSER

FTCGCC integrates third party software Java Document Object Model (JDOM). JDOM was created by Jason Hunter and Brett. JDOM is a method of representing XML document for easy reading/writing and manipulation. JDOM is an open source initiative with Apache style license. JDOM integrates with Document Object Model (DOM) and Simple API for XML (SAX).

JDOM salient features include

- It is light weight
- It can represent full document
- It supports document modification
- It is easy to use

FTCGCC imports Application Programming Interfaces (APIs) of following four classes

- org.jdom.Document
- org.jdom.Element
- org.jdom.output.XMLOutputter
- org.jdom.output.Format

10.8.3 Data structures and functions

Java's ArrayList can dynamically grow or shrink and is a variable length array of object references. Vector is similar to ArrayList, but is synchronized. Table 14. gives the important data structures and functions. Description column details the intended functionality.

Table 17. Important data structures and functions

Entity	Description
private Vector<Mote> motes	Stores the motes in Vector
private Vector<MoteType> moteTypes	Stores the moteTypes in Vector
public static ArrayList<Element> config	Stores the XML elements in ArrayList
public MoteType[] getMoteTypes()	Returns all mote types in simulation
public MoteType getMoteType(String identifier)	Returns mote type with given identifier.
public Collection<Element> getConfigXML()	Returns the current simulation config represented by XML elements.
public boolean setConfigXML()	Sets the current simulation config depending on the given configuration
public void saveSimulationConfig(File file)	Saves the Simulation Config as an XML

IJSER

10.9 FTCGCC usage in Contiki environment

The tool usage is summarized in the following steps.

1) Go to the github https://github.com/Abhinandan1414/CoojaTestCaseGeneration", download the content
2) Create an input file with the syntax which adheres to syntax of "GenTest.txt"
3) Set the CLASSPATH to $CLASSPATH:cooja.jar:.
4) Copy the artifacts to directory of your setup

IJSER © 2019
http://www.ijser.org

5) Create directory lib and copy jdom.jar jsyntaxpane.jar log4j.jar JDOM_LICENSE JSYNTAXPANE_LICENSE LOG4J_LICENSE
6) Then compile the GenTestcsc.java
7) Run GenTestcsc

We successfully used the tool FTCGCC for generating additional test cases to be added to the base regression test suite of Contiki and Cooja and saved approximately 100 lines of XML coding per test case. Since we added additional 35 test cases, we saved approximately 3500 lines of XML code. A sample XML file generated is depicted in the Figure 6.
Complete code to be used along with the usage details is in Git hub repository [37]. Final logs of the test cases are kept in the Google docs repository [32].

10.10 Conclusion

In this work we present our tool Functional Test Case Generator for Contiki and Cooja. We have successfully used the tool to generate functional test cases. We looked at the augmentation of the existing regression test suite. Using our tool, a user could customize the number of test cases which are added to the existing regression test suite. This will result in a customized generation with the requirement of code coverage, execution time etc. Use of our proposed tool will reduce the manual effort of manual coding needed for the functional test case generation.

IJSER © 2019
http://www.ijser.org

11 Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques

11.1 Introduction

In this chapter test suite prioritization using residual coverage algorithm for white box testing is discussed. For black box testing a new statistical technique is introduced in this chapter. A mechanism to solve the breaking of tie in residual test coverage algorithm is also presented in this chapter. Further new metric is introduced for prioritizing in the black box testing.

In a typical development cycle software professional works in multiple capacities. Missing exposure to either development or testing is partial view of the development life cycle [8]. The tester without the development cycle knowledge can be risk during the white box testing and developer without testing knowledge means quality compromise.

Regression test selection, minimization and prioritization are the main topics of research in several papers. There are two main types of coverage.

- Requirement coverage: In this approach test cases to requirement tracing happens to ensure that all the requirements are covered and requirement specification document serves as test oracle.
- Code coverage: In this approach statement coverage, branch coverage and loop coverage are measured to ensure that testing has not missed executing the code.

We use code coverage as a measurement criteria both for white box and black box testing.

11.2 Test Coverage Algorithm for White Box Testing

For a given software to be tested, consider P to be the product code before bug fixes and enhancement and P’ the product code after the modification. The corresponding test suites are denoted by T and T’ respectively. Now, one way

IJSER © 2019
http://www.ijser.org

to visualize T is as collection of obsolete, redundant and valid test cases. In other words T is collection of {obsolete, redundant, valid} test cases. The T' will encompass {valid, newly added test cases}.

There are different schools of thoughts when it comes to selecting regression tests between successive releases. Each with their own set of advocates. As Aditya P. Mathur aptly classifies [8], the philosophy for each of these categories is very different. The categories are:

- Test all: Brute force method with long execution cycles. Most widely used technique in the commercial world combined with automation of regression testing
- Random selection: Sampling of the test cases. Better than no regression testing at all. Test cases are picked from the test suite except obsolete test cases randomly.
- Selecting modification traversing tests: Assumes the tester to have the knowhow of the complete product code. Better than test all and random selection process. Usually the testers classify the test cases in buckets where they put related test cases in a given bucket. Depending upon how the test cases are chosen this mode of testing may or may not yield good results.
- Test minimization: Pruning of the test suite by dropping the test cases with similar product trace code as they are redundant. This results in new reduced size of the regression test suite.
- Test Prioritization: Assumes that the test cases can be ranked on the basis of certain criteria.

It is the last school of thought which is the focus of this chapter. Regression test suite prioritization is necessary when there is crunch of resources and time.

Large organizations may choose to use the combination of all the approaches mentioned above. Large execution cycle may mean drain on resources and time and therefore good amount of research has gone in optimizing the regression test suites.

The remaining sections of the chapter can be broadly classified into the following sections.

- Residual test coverage algorithm enhancements for regression test suite prioritization using white box testing.
- Statistical techniques for regression test suite prioritization using black box testing.
- Process flow at the implementation level which can aid the above two processes.

IJSER © 2019
http://www.ijser.org

- Case study of coverage tool to explain how metrics of choice can be extracted.

If the current test suite is visualized as the collection of {obsolete, redundant, valid test cases}, the test suite for new version will be {valid test cases, newly added test cases}. The newly added test cases either cover newly added functionality or they cover modified functionality. In this case some of the valid test cases may become redundant or obsolete. If the removed functionality is added back the obsolete test cases may become valid test cases. The new test cases go through the review process depending upon review process in place. The newly added test cases always go through review process and are always executed. The remaining test cases need to be prioritized. Modified test cases can be considered as new test cases. Between successive releases the historical data needs to be preserved. The coverage data and execution time becomes the reference data for new build of the product. The newly added test cases are always executed and ranking them among themselves adds little value. The test execution will be newly added test cases followed by prioritized test cases. Since the newly added test cases will be far less compared to valid test cases taken from previous releases the approaches mentioned in this chapter should work in practical setup. There is no real gain in ranking the newly added test cases as they are mandatorily executed.

11.3 Residual Test Coverage Algorithm enhancements for White Box Testing

Here we present an algorithm for breaking the tie which is required when dealing with large multiparameter software.
We modify the residual test coverage algorithm to ensure that the random test selection process is not used. Here, newly introduced parameters are in italics and modified algorithm looks as follows.

Algorithm for prioritizing the regression test suite post modification.

Input T’: Set of regression tests for the modified program P’.

entitiesCov: Set of entities in P covered by tests in T’.

cov: Coverage vector such that for each test t Ɛ T’, cov(t) is the set of entities covered by executing P against t.

IJSER © 2019
http://www.ijser.org

executionTime: executionTime(t) is the time taken by the test case t Ɛ T' to complete the execution.

linesOfCodeTraced: linesOfCodeTraced(t) is the total lines of product code covered by the test case t Ɛ T' during execution.

Output PrT: A sequence of prioritized test cases such that

(a) each test case belongs to T'

(b) each test in T' appears exactly once in PrT, and

(c) tests in PrT are arranged in the ascending order of cost.

Step 1: X' = T'. Find t Ɛ X' such that |cov(t)|≥ |cov(u)|for all u Ɛ X', u ≠ t.

Step 2: Set PrT = <t>, X' = X'\{t}. Update entitiesCov by removing from it all entities covered by t. Thus entitiesCov=entitiesCov\cov(t).

Step 3: Repeat the following steps while X' ≠ Φ and entityCov ≠ Φ.

Step 3.1 Compute the residual coverage for each test t Ɛ T'. resCov(t) = |entitiesCov \ (cov(t) ∩entitiesCov |. resCov(t) indicates the count of currently uncovered entities that will remain uncovered after having executed P against t.

Step 3.2 Find test t Ɛ X' such that resCov(t) ≤ resCov(u), for all u Ɛ X', u ≠ t. *If two or more such tests exist then first compare |cov(t)|, if there is tie again look for executionTime(t) and linesOfCodeTraced(t) and select the one with high |cov(t)|, linesOfCodeTraced(t) and least executionTime(t).*

Step 3.3 Update the prioritized sequence, set of tests remaining to be examined, and entities yet to be covered by tests in PrT. PrT = append(PrT, t), X'=X'\{t}, and entitiesCov = entitiesCov\cov(t).

Step 4: Append to PrT any remaining tests n X'. All remaining tests have the same residual coverage which equals |entitiesCov|. *Hence these tests are tied. Now follow exactly what was done in step3.2. That is when two or more tests tie look for test case with higher value of |cov(t) |and linesOfCodeTraced(t) and least executionTime(t).*

End of Algorithm

IJSER © 2019
http://www.ijser.org

In the proposed algotirhm, to begin with, we start with test case that covers maximum entities. Then we choose the test case which will run the maximum uncovered entities that are not already covered by previously run test cases. The second step is repeated till there are no more unique entities to be covered. When we encounter two or more test cases with the same cost, we choose the test case that will cover maximum entities with high loc trace and least execution time. Once all the entities are covered, we choose the test cases with the same criteria, high entities coverage, least execution time and high loc trace. The logic is, choose the test case that covers maximum entities, traces more loc in least time.

This is a methodology which comes under white box testing. In an ideal situation, a tester carries out the testing with a member of the of the team of developers.

11.4 Statistical Approach for Prioritization of Test Cases for Black Box Testers.

The approach mentioned in the Section 4.2 involves product code dissection at class function level. This requires an understanding of the code being handled. In situations where there is lack of time for understanding and implementing a white box algorithms, black box techniques can be used. As will be explained in further sections, this can be done with probes in the test cases and tweaking of the tool used for gathering the code coverage data.

Please have look at the table below.

IJSER

Table 18. Table for calculating the new metric

Test case number	Number of lines of code in the test case (LOCTesti)	Number of lines of code traced in the product code(LOCProdi)	Execution time (τi)	New metric (Nmi)
1				
2				
. .				
n				
Effectiveness of total test suite = $\sum_{i=1}^{i=n} \mathrm{Nmi}$				

IJSER © 2019
http://www.ijser.org

$$\text{Effectiveness of test suite per test case} = \frac{\sum_{i=1}^{i=n} \text{Nmi}}{n}$$

Where $\text{Nmi} = \frac{\text{LOCProdi}}{(\text{LOCTesti X } \tau\text{i})}$

Here we are using the logic that the test case which traces maximum lines of product code with least number of lines in itself in least amount of time is efficient. We sum up the numbers Nmi short for New Metric to get the picture at the test suite level.

However the above explained logic assumes there are no redundant test cases in test suite.

11.5 Coverage Tools: CodeCover a case study

Since we have used the terms such as product lines of code traversed and lines of the code in the test case we need a tool to measure the same. The obvious choice is code coverage tools. There are plenty of coverage tools for code coverage. The choice of tool for particular project depends upon various criteria. We find that the tool CodeCover is most suited tool for Java projects since it comes with EPL license and most importantly, it is open source software. It is backed by a team of developers who support the tool usage and deal with any issues. The tool can be extended and tweaked as per the needs of particular project. CodeCover gives various metrics such as statement coverage, branch coverage, loop coverage, MC/DC coverage etc at test case level as well as test suite level. There are provisions to call the Application Programmers Interface (APIs) of this tool from outside. The flexibility to call the APIs from external environment can mean a lot to implement the topics explained in this chapter.

In further sections we explain how the APIs could be called from test setup to gather the required data and how it can be combined with other parameter of interest viz. execution time.

The CodeCover already supports two languages as diverse as Java and Cobol. This is being mentioned for reason. That is tweaking of the tool is not effort intensive.

IJSER © 2019
http://www.ijser.org

11.6 Process Flow for Collecting Metrics of Choice

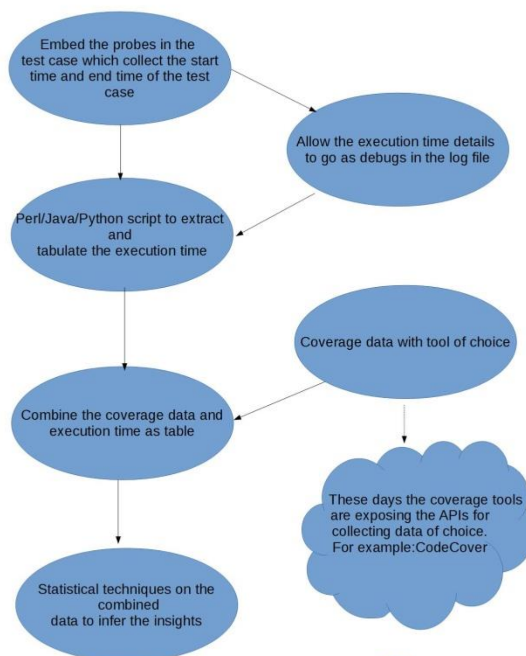


Figure 27. Process flow for collecting the metrics of choice

Figure 27. details how the product code traced and execution time of the test cases are collected while gathering the metrics of choice.

11.7 Advantages of Test Suite Prioritization

The advantages of spending effort in test suite prioritization are best seen in projects which maintains thousands of test cases. Further, depending upon the system, it could take few days to execute the test cases. These kinds of systems are seen in many of the large software developments. These occur for large scale development of free software in wireless networking related area.

IJSER © 2019
http://www.ijser.org

11.7 Conclusion

This chapter explains two approaches black box testing and white box testing. Each of the approach has its own merits and drawbacks. Depending upon the resources and time either approach can be followed.

The chapter also explains how the metrics being discussed in either approach can be extracted in practical setup. The CodeCover based approach is discussed to explain how the metrics of choice can be extracted.

IJSER

IJSER © 2019
http://www.ijser.org

12. Conclusion

This chapter is conclusion chapter of the Thesis.

12.1 Introduction: The research problem

This Thesis studies the following research problems.

- Study of regression test suites in general
- Design and implementation of combinatorial testing based test suites for internet of things operating system and its simulators
- Measuring the effectiveness of designed test suites using the traditional coverage techniques like code coverage
- Automation of test scripts generation from combinatorial testing design model and analyzing coverage to refine the combinatorial testing design model.
- Propose an integrated test environment for combinatorial testing

12.1.1 Summary of results

The summary of the results are as shown in the table 10 below. Table 10 gives traceability between research problem and outcome

Table 19. Traceability from research problems to the outcomes

Research Problem	**Traceability to Thesis chapter**
Study of regression test suites with respect to execution time and residual test coverage algorithm enhancement.	Chapter 3 and 11
Design and implementation of combinatorial testing based test suites for internet of things operating system and its simulators	Chapter 5,7 and 9
Measuring the effectiveness of designed test suites using the traditional coverage techniques like code coverage	Chapter 9
Automation of test scripts generation from combinatorial testing design model and analyzing coverage to refine the combinatorial testing design model.	Chapter 9
Propose an integrated test environment for combinatorial testing	Chapter 4

IJSER © 2019
http://www.ijser.org

Demonstration of CT-RTS	Chapter 6

12.2 Conclusions.

Based on the Thesis contribution following conclusions can be drawn.

1. Statistical techniques can be applied to study the execution time of a generic regression test suite. It is possible to study the statistical parameters such as probability distribution function, correlation etc of regression test suites. Further using statistical techniques it is possible to interpolate or extrapolate the execution time of test suites which will help during the pruning or augmentation of the test suite.

2. It is possible to prioritize the regression test suite using the residual test coverage algorithm which takes the code coverage and execution time as the input parameter. Further, black box approach to test suite prioritization using statistical techniques is possible.

3. Integrated test environment approach brings the better integration of different tools for CT. It is a centralized approach which reduces the number of tools and duplicated functionality. Maintenance becomes simple.

4. Using ACTS tool it is possible to come up with an effective test suite for a multi parameter software. The effectiveness of the test suite is cross verified using the traditional code coverage metrics.

5. It is possible to come up with a test design methodology for Internet of Things operating system using the combinatorial approach.

6. Re-architecture of regression test suite using CT approach is possible. Contiki OS and Cooja simulator regression test suites are used as case studies. The effectiveness of the test suites designed using CT is cross verified using the traditional metrics of code coverage using the tools such as CodeCover and Clover.

IJSER © 2019
http://www.ijser.org

12.2 Future work

Following are the scopes for future works:

1) CT-RTS is applied for Contiki and its simulator Cooja. Similar techniques can be explored with the other operating systems such as RIOT and TinyOS. At the time of this thesis writing there were no readily available simulators for the operating systems such as RIOT and TinyOS. Since the time required to develop the simulators for these operating systems was significant it is set aside as future work.
2) CT-RTS can be used for other open source softwares.
3) Integrated test environment can be explored for real life project with the commercial tools mentioned in the thesis.
4) The statistical techniques mentioned in this thesis can be explored further and more rigorously.
5) Test suite prioritization using the Residual test coverage algorithm can be explored more rigorously.

IJSER

IJSER © 2019
http://www.ijser.org

Appendix A: ACTs Generated Test Design for Contiki Operating System.

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10
Test Case#	Platform	base	Rime	NetPerformanc	collect	ipv4	ipv6	RPL	ipv6apps
0	Exp5438	coffee	rucb	NetPerf-lpp	shell-collect-lossy	webserver	udp-fragmentation	up-root	coap
1	Exp5438	checkpointing	deluge	NetPerf-cxmac	shell-collect	telnet-ping	unicast-fragmentatio	root-reboot	servreg-hack
2	Exp5438	Multithreading	runicast	NetPerf	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	large-network	coap
3	Exp5438	coffee	trickle	NetPerf-cxmac	shell-collect	webserver	ipv6-udp	upanddownroutes	servreg-hack
4	Exp5438	checkpointing	mesh	NetPerf	shell-collect	webserver	udp-fragmentation	temporaryrootloss	coap
5	Exp5438	Multithreading	collect	NetPerf-lpp	shell-collect	webserver	unicast-fragmentatio	randomrearrngement	servreg-hack
6	Exp5438	checkpointing	rucb	NetPerf-cxmac	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	rpl-dao	servreg-hack
7	z1	Multithreading	deluge	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	up-root	coap
8	z1	coffee	runicast	NetPerf-lpp	shell-collect	webserver	udp-fragmentation	root-reboot	servreg-hack
9	z1	checkpointing	trickle	NetPerf-lpp	shell-collect-lossy	telnet-ping	unicast-fragmentatio	large-network	coap
10	z1	Multithreading	mesh	NetPerf-cxmac	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	upanddownroutes	coap
11	z1	coffee	collect	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	temporaryrootloss	servreg-hack
12	z1	checkpointing	rucb	NetPerf	shell-collect	telnet-ping	ipv6-udp	randomrearrngement	coap
13	z1	Multithreading	deluge	NetPerf-lpp	shell-collect	webserver	udp-fragmentation	rpl-dao	coap
14	wismote	checkpointing	runicast	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatio	up-root	servreg-hack
15	wismote	Multithreading	trickle	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	root-reboot	coap
16	wismote	coffee	mesh	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	large-network	servreg-hack
17	wismote	checkpointing	collect	NetPerf-cxmac	shell-collect-lossy	telnet-ping	udp-fragmentation	upanddownroutes	coap
18	wismote	Multithreading	rucb	NetPerf	shell-collect-lossy	telnet-ping	unicast-fragmentatio	temporaryrootloss	coap
19	wismote	coffee	deluge	NetPerf-lpp	shell-collect-lossy	webserver	ipv6-rpl-collect	randomrearrngement	servreg-hack
20	wismote	coffee	runicast	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	rpl-dao	coap
21	micaz	checkpointing	trickle	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	up-root	servreg-hack
22	micaz	coffee	mesh	NetPerf-lpp	shell-collect-lossy	telnet-ping	unicast-fragmentatio	root-reboot	coap
23	micaz	Multithreading	collect	NetPerf-cxmac	shell-collect	webserver	ipv6-rpl-collect	large-network	coap
24	micaz	checkpointing	rucb	NetPerf	shell-collect	telnet-ping	ipv6-udp	upanddownroutes	coap
25	micaz	checkpointing	deluge	NetPerf-lpp	shell-collect-lossy	webserver	ipv6-rpl-collect	temporaryrootloss	servreg-hack
26	micaz	checkpointing	runicast	NetPerf-cxmac	shell-collect	telnet-ping	udp-fragmentation	randomrearrngement	servreg-hack
27	micaz	checkpointing	trickle	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatio	rpl-dao	servreg-hack

IJSER

28	sky	coffee	mesh	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
29	sky	checkpointing	collect	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
30	sky	Multithreading	rucb	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	large-network	servreg-hack
31	sky	checkpointing	deluge	NetPerf-lpp	shell-collect-lossy	telnet-ping	unicast-fragmentatio	upanddownroutes	servreg-hack
32	sky	checkpointing	runicast	NetPerf-cxmac	shell-collect	telnet-ping	unicast-fragmentatio	temporaryrootloss	coap
33	sky	checkpointing	trickle	NetPerf-lpp	shell-collect	webserver	ipv6-rpl-collect	randomrearrngement	coap
34	sky	coffee	mesh	NetPerf-lpp	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	rpl-dao	coap
35	jcreate	coffee	collect	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
36	jcreate	checkpointing	rucb	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
37	jcreate	Multithreading	deluge	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	large-network	coap
38	jcreate	Multithreading	runicast	NetPerf-lpp	shell-collect	webserver	unicast-fragmentatio	upanddownroutes	servreg-hack
39	jcreate	checkpointing	trickle	NetPerf-lpp	shell-collect-lossy	webserver	ipv6-udp	temporaryrootloss	coap
40	jcreate	coffee	mesh	NetPerf-cxmac	shell-collect	telnet-ping	ipv6-udp	randomrearrngement	servreg-hack
41	jcreate	checkpointing	collect	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatio	rpl-dao	servreg-hack
42	sentilla-usb	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
43	sentilla-usb	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
44	sentilla-usb	Multithreading	runicast	NetPerf-cxmac	shell-collect-lossy	webserver	udp-fragmentation	large-network	coap
45	sentilla-usb	coffee	trickle	NetPerf	shell-collect-lossy	webserver	unicast-fragmentatio	upanddownroutes	coap
46	sentilla-usb	coffee	mesh	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	temporaryrootloss	coap
47	sentilla-usb	Multithreading	collect	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	randomrearrngement	coap
48	sentilla-usb	Multithreading	collect	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatio	rpl-dao	servreg-hack
49	esb	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
50	esb	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
51	esb	Multithreading	runicast	NetPerf-cxmac	shell-collect-lossy	webserver	udp-fragmentation	large-network	servreg-hack
52	esb	Multithreading	trickle	NetPerf	shell-collect-lossy	webserver	unicast-fragmentatio	upanddownroutes	servreg-hack
53	esb	coffee	mesh	NetPerf-lpp	shell-collect	webserver	ipv6-rpl-collect	temporaryrootloss	servreg-hack
54	esb	Multithreading	collect	NetPerf	shell-collect	webserver	ipv6-udp	randomrearrngement	coap
55	esb	coffee	trickle	NetPerf-cxmac	shell-collect	telnet-ping	udp-fragmentation	rpl-dao	servreg-hack
56	native	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
57	native	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
58	native	Multithreading	runicast	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	large-network	servreg-hack
59	native	coffee	trickle	NetPerf-lpp	shell-collect-lossy	webserver	unicast-fragmentatio	upanddownroutes	coap
60	native	coffee	mesh	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatio	temporaryrootloss	servreg-hack
61	native	Multithreading	collect	NetPerf-cxmac	shell-collect	telnet-ping	ipv6-udp	randomrearrngement	coap
62	native	checkpointing	rucb	NetPerf-cxmac	shell-collect-lossy	webserver	ipv6-rpl-collect	rpl-dao	coap
63	cooja	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack

IJSER

64	cooja	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
65	cooja	Multithreading	runicast	NetPerf-cxmac	shell-collect	telnet-ping	udp-fragmentation	large-network	servreg-hack
66	cooja	Multithreading	trickle	NetPerf	shell-collect-lossy	telnet-ping	unicast-fragmentatio	upanddownroutes	servreg-hack
67	cooja	coffee	mesh	NetPerf-cxmac	shell-collect-lossy	telnet-ping	ipv6-udp	temporaryrootloss	coap
68	cooja	Multithreading	collect	NetPerf-lpp	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	randomrearrngement	coap
69	cooja	checkpointing	collect	NetPerf	shell-collect	webserver	udp-fragmentation	rpl-dao	coap

IJSER

IJSER © 2019
http://www.ijser.org

Appendix B: Code Coverage Data Gathered for Existing Test Suite of Contiki and Cooja using CodeCover

up to the title page

se . sics . cooja

	Statement Coverage		Branch Coverage		Loop Coverage		Strict Condition Coverage	
contikimote	102 / 747	13 %	40 / 323	12 %	7 / 99	7 %	11 / 191	5 %
dialogs	162 / 2130	7 %	45 / 907	4 %	15 / 225	6 %	6 / 483	1 %
interfaces	79 / 421	18 %	40 / 178	22 %	8 / 48	16 %	10 / 100	10 %
plugins	327 / 4875	6 %	83 / 2091	3 %	16 / 620	2 %	19 / 1127	1 %
motes	11 / 148	7 %	6 / 80	7 %	0 / 27	0 %	4 / 43	9 %
util	14 / 379	3 %	11 / 256	4 %	3 / 114	2 %	3 / 145	2 %
radiomediums	169 / 353	47 %	85 / 195	43 %	53 / 120	44 %	31 / 115	26 %
emulatedmote	16 / 135	11 %	0 / 52	0 %	0 / 3	0 %	0 / 26	0 %
positioners	0 / 125	0 %	0 / 38	0 %	0 / 0	---	0 / 19	0 %
HasQuickHelp	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
MoteInterfaceHandler	19 / 42	45 %	14 / 46	30 %	3 / 33	9 %	6 / 23	26 %
CoreComm	8 / 57	14 %	6 / 32	18 %	1 / 18	5 %	0 / 10	0 %
Plugin	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
RadioMedium	1 / 1	100 %	0 / 0	---	0 / 0	---	0 / 0	---
ContikiError	0 / 3	0 %	0 / 0	---	0 / 0	---	0 / 0	---
GUI	357 / 1259	28 %	211 / 787	26 %	43 / 207	20 %	45 / 419	10 %
ConvertedRadioPacket	2 / 2	100 %	0 / 0	---	0 / 0	---	0 / 0	---
COOJARadioPacket	0 / 1	0 %	0 / 0	---	0 / 0	---	0 / 0	---
AbstractionLevelDescription	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
MoteMemory	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
RadioConnection	19 / 28	67 %	4 / 14	28 %	0 / 0	---	0 / 7	0 %
SectionMoteMemory	0 / 70	0 %	0 / 24	0 %	0 / 21	0 %	0 / 17	0 %
COOJAProject	8 / 21	38 %	1 / 28	3 %	0 / 6	0 %	0 / 16	0 %
RadioPacket	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
AddressMemory	0 / 1	0 %	0 / 0	---	0 / 0	---	0 / 0	---
SimEventCentral	46 / 68	67 %	17 / 36	47 %	11 / 45	24 %	4 / 22	18 %

IJSER © 2019
http://www.ijser.org

Appendix C: Tweaking of Ant build.xml for Gathering The Coverage Data with CodeCover

```
<?xml version="1.0"?>
<project name="COOJA Simulator" default="run" basedir=".">
<property name="java" location="java"/>
.
.
<property
name="args"
value="/>
<property
name="codecover
Dir"
value="/home/user/Desktop/CodeCover/codecover-batch-1.0/lib"/>
<property name="sourceDir" value="/home/user/contiki-2.7/tools/cooja/java"/>
<property name="instrumentedSourceDir" value="instrumented"/>
<property name="mainClassName" value="se.sics.cooja.GUI"/>
<taskdef name="codecover" classname="org.codecover.ant.CodecoverTask"
classpath="${codecoverDir}/codecover-ant.jar"/>
<target name="clean">
<delete>
<fileset dir="." includes="*.clf"/>
</delete>
<delete file="codecover.xml"/>
<delete file="report.html"/>
<delete dir="report.html-files"/>
</target>
<target name="instrument-sources" depends="clean">
<codecover>
<instrument containerId="c"
language="java"
destination="${instrumentedS
ourceDir}" charset="utf-8"
copyUninstrumented="yes">
<source dir="${sourceDir}">
<include name="**/*.java"/>
</source>
</instrument>
<save containerId="c" filename="codecover.xml"/>
</codecover>
</target>
<target name="compile-instrumented"
depends="instrument-sources"> <javac
srcdir="${instrumentedSourceDir}"
destdir="${instrumentedSourceDir}" encoding="utf-
8" target="1.7" debug="true"
```

IJSER

IJSER © 2019
http://www.ijser.org

```
classpath="${codecoverDir}/lib/codecover-
instrumentationjava.
jar:/home/user/contiki-2.7/tools/cooja/lib/log4j.jar:/home/user/contiki-
2.7/tools/cooja/lib/jdom.jar:/home/user/contiki-
2.7/tools/cooja/lib/jsyntaxpane.jar"
includeAntRuntime="false"></javac> </target>
<target name="run-instrumented" depends="compile-
instrumented, copy configs"> <java
classpath="${instrumentedSourceDir}:${codecoverDir}/lib/codec
overinstrumentation- java.jar:/home/user/contiki-
2.7/tools/cooja/lib/log4j.jar:/home/user/contiki-
2.7/tools/cooja/lib/jdom.jar:/home/user/contiki-
2.7/tools/cooja/lib/jsyntaxpane.jar" fork="true" failonerror="true"
classname="${mainClassName}">
<jvmarg value="-Dorg.codecover.coverage-log-file=test.clf"/>
</java>
</target>
<target name="create-report" >
<codecover>
<load containerId="c" filename="codecover.xml"/>
<analyze containerId="c" coverageLog="*.clf" name="Test Session"/>
<save containerId="c"
filename="codecover.xml"/>
<report containerId="c"
destination="report.html"
template="/home/user/Desktop/CodeCover/codecover-batch-1.0/reporttemplates/
HTML_Report_hierarchic.xml">
<testCases>
<testSession pattern=".*">
<testCase pattern=".*"/>
</testSession>
</testCases>
</report>
</codecover>
</target>
<target name="help">
<echo>
.
.
<target name="copy configs" depends="init">
<mkdir dir="${build}"/>
<copy todir="/home/user/contiki-
2.7/tools/cooja/instrumented"> <fileset
dir="${config}"/>
</copy>
.
.
.
<target name="jar_cooja" depends="init, compile, copy
configs, compile instrumented ">
<mkdir dir="${dist}"/>
<jar destfile="${dist}/cooja.jar" base dir="/home/user/contiki-
2.7/tools/cooja/instrumented">
<manifest>
```

IJSER

```
        <attribute name="Main-Class"
        value="se.sics.cooja.GUI"/> <attribute
        name="Class-Path" value=". lib/log4j.jar
        lib/jdom.jar lib/jsyntaxpane.jar"/>
        </manifest>
        </jar>
        <mkdir dir="${dist}/lib"/>
        <copy todir="${dist}/lib">
        <fileset dir="${lib}"/>
        </copy>
        </target>
</project>
```

IJSER

IJSER © 2019
http://www.ijser.org

APPENDIX D: ACTS Test Design Input for Re-engineered Test Suite

Parameters:

Platform	[Exp5438, z1, wismote, micaz, sky, jcreate, sentilla-usb, esb, native, cooja]
base	[Multithreading, coffee, checkpointng, none]
Rime	[collect, rucb, deluge, runicast, trickle, mesh, none]
NetPerformance	[NetPerf, NetPerf-lpp, NetPerf-cxmac, none]
collect	[shell-collect, shell-collect-lossy, none]
ipv4	[telnet-ping, webserver, none]
ipv6	[ipv6-udp, udp-fragmentation, unicast-fragmentation, ipv6-rpl-collect, none]
RPL	[up-root, root-reboot, large-network, upanddownroots, temporaryrootloss, randomrearrangement, rpl-dao, none]
ipv6apps	[servreg-hack, coap, none]

IJSER

Relations:

Constraints :

(base != "none") => (Rime == "none")
(base != "none") => (NetPerformance == "none")
(base != "none") => (collect == "none")
(base != "none") => (ipv4=="none")
(base != "none") => (ipv6=="none")
(base != "none") => (RPL =="none")
(base != "none") => (ipv6apps == "none")
(Rime != "none") => (base=="none")
(Rime != "none") => (NetPerformance == "none")
(Rime != "none") => (collect == "none")
(Rime != "none") => (ipv4 == "none")
(Rime != "none") => (ipv6 == "none")
(Rime != "none") => (RPL == "none")
(Rime != "none") => (ipv6apps == "none")
(NetPerformance != "none") => (base == "none")
(NetPerformance != "none") => (Rime == "none")

IJSER © 2019
http://www.ijser.org

(NetPerformance != "none") => (collect == "none")
(NetPerformance != "none") => (ipv4 == "none")
(NetPerformance != "none") => (ipv6 == "none")
(NetPerformance != "none") => (RPL == "none")
(NetPerformance != "none") => (ipv6apps == "none")
(collect != "none") => (base == "none")
(collect != "none") => (Rime == "none")
(collect != "none") => (NetPerformance == "none")
(collect != "none") => (ipv4 == "none")
(collect != "none") => (ipv6 == "none")
(collect != "none") => (RPL == "none")
(collect != "none") => (ipv6apps == "none")
(ipv4 != "none") => (base == "none")
(ipv4 != "none") => (Rime == "none")
(ipv4 != "none") => (NetPerformance == "none")
(ipv4 != "none") => (collect == "none")
(ipv4 != "none") => (ipv6 == "none")
(ipv4 != "none") => (RPL == "none")
(ipv6 != "none") => (base == "none")
(ipv6 != "none") => (Rime == "none")
(ipv6 != "none") => (NetPerformance == "none")
(ipv6 != "none") => (collect == "none")
(ipv6 != "none") => (ipv4 == "none")
(ipv6 != "none") => (RPL == "none")
(ipv6 != "none") => (ipv6apps == "none")
(RPL != "none") => (base == "none")
(RPL != "none") => (Rime == "none")
(RPL != "none") => (NetPerformance == "none")
(RPL != "none") => (collect == "none")
(RPL != "none") => (ipv4 == "none")
(RPL != "none") => (ipv6 == "none")
(RPL != "none") => (ipv6apps == "none")
(ipv6apps != "none") => (base == "none")
(ipv6apps != "none") => (Rime == "none")
(ipv6apps != "none") => (NetPerformance == "none")
(ipv6apps != "none") => (collect == "none")
(ipv6apps != "none") => (ipv4 == "none")
(ipv6apps != "none") => (ipv6 == "none")
(ipv6apps != "none") => (RPL == "none")
(base != "none") || (Rime != "none") || (NetPerformance != "none") || (collect !=

IJSER © 2019
http://www.ijser.org

"none") || (ipv4 != "none") ||(ipv6 != "none") ||
(RPL != "none") || (ipv6apps != "none")

IJSER

IJSER © 2019
http://www.ijser.org

APPENDIX E: ACTS Test Design for Cooja Test Suite

Input Parameter Model 1:

Parameters:

FileOperation [NewSimulation, OpenSimulation, CloseSimulation, SaveSimulation, ExportSimulation, Exit]

Simulation [StartSimulation, ReloadSimulation, ControlPanel, Simulation, Null]

Motes [AddMotes, MoteTypes, RemoveAllMotes, Null]

Relations:
[2,(Simulation, Motes)]
Constraints :
(FileOperation = "CloseSimulation") => (Simulation == "Null")
(FileOperation = "CloseSimulation") => (Motes == "Null")
(FileOperation = "Exit") => (Simulation == "Null")
(FileOperation = "Exit") => (Motes == "Null")

IJSER

Input Parameter Model 2:

Parameters:

FileOperation [NewSimulation, OpenSimulation, CloseSimulation, SaveSimulation, ExportSimulation, Exit]

Simulation [StartSimulation, ReloadSimulation, Null]

Tools [Network, MoteOutPut, TimeLine, BreakPoints, RadioMessages, SimulationScriptEditor, Notes, BufferView, MoteRadioDutyCycle, MoteInformstion, MoteInterfaceViewer, VariableWatcher,MSPCli, MSPCodeWatcher, MSPStackWatcher, MSPCycleWatcher, SerialSocket, CollectView, Null]

Relations:
[2,(Simulation, Tools)]

IJSER © 2019
http://www.ijser.org

Constraints :
(FileOperation = "CloseSimulation") => (Simulation=="Null")
(FileOperation = "CloseSimulation") => (Tools == "Null")
(FileOperation = "Exit") => (Simulation == "Null")
(FileOperation = "Exit") => (Tools == "Null")
(FileOperation = "SaveSimulation") => (Simulation=="Null")
(FileOperation = "SaveSimulation") => (Tools == "Null")
(FileOperation = "ExportSimulation")=>(Simulation=="Null")
(FileOperation = "ExportSimulation") => (Tools == "Null")

Input Parameter Model 3:

Parameters:

FileOperation	[NewSimulation]
RadioMedium	[UDGM_DistanceLoss, UDGMConstantloss, DirectedGraphRadioMedium, NoRadioTraffic, MutiPathRayTraceMedium]
CreateNewMoteType	[DisturberMote, ImportJavaMote, CoojaMote, MicazMote, SkyMote, Exp430F5438Mote, Wismote, Z1Mote]
Tools	[Network, MoteOutput, TimeLine, BreakPoints, RadioMessages, SimulationScriptEditor, BufferView, MoteRadioDutyCycle, MoteInformation, MoteInterfaceViewer, VariableWatcher, MSPCli, MSPCodeWatcher, MSPStackWatcher, SerialClientSocket, SerialServerSocket, CollectView]

Relations:
[3,(RadioMedium, CreateNewMoteType, Tools)]

Input Parameter Model 4:

Parameters:

FileOperation	[OpenSimulation]
IOTScenario	[RplUdp, RplUdpPowerTrace, SkyWebSense, UnicastExample, BroadCastExample,RplCollectTreeDenseNoloss, RplCollectTreeSparseLossy, UdpStream, TrickleLibrary,RimeCollect, RimeBroadCast, HelloWorld, Netdb,

IJSER © 2019
http://www.ijser.org

	NetPerfSky, ServerClient, ServerOnly, CoapServerClientExample, RestServerExample]
Tools	[Network, MoteOutPut, TimeLine, BreakPoints, RadioMessages, SimulationScriptEditor,BufferView, MoteRadioDutyCycle, MoteInformation, MoteInterfaceViewer, VariableWatcher,MSPCli, MSPCodeWatcher, MSPStackWatcher, SerialClientSocket,SerialServerSocket, Collectview]

Relations:

IJSER

IJSER © 2019
http://www.ijser.org

Appendix F: Code for Auto Generating csc Files.

```
import se.sics.cooja.*;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Observable;
import java.util.Observer;
import java.util.Random;
import java.util.Vector;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.XMLOutputter;
import org.jdom.output.Format;

import java.io.FileOutputStream;
import java.util.zip.GZIPOutputStream;
import java.io.OutputStream;
import java.io.File;
import java.io.IOException;

import
se.sics.cooja.VisPlugin.PluginRequiresVisualizationException
;

import javax.swing.JDesktopPane;
import javax.swing.JInternalFrame;

import java.util.regex.*;
import java.nio.file.*;


public class GenTestcsc {
      public static final long MICROSECOND = 1L;
      public static final long MILLISECOND =
1000*MICROSECOND;

      /*private static long EVENT_COUNTER = 0;*/

      private Vector<Mote> motes = new Vector<Mote>();
      private Vector<Mote> motesUninit = new Vector<Mote>();

      private Vector<MoteType> moteTypes = new
Vector<MoteType>();
```


IJSER © 2019
http://www.ijser.org

```
      /* If true, run simulation at full speed */
      private boolean speedLimitNone = true;
      /* Limit simulation speed to maxSpeed; if maxSpeed is
1.0 simulation is run at real-time speed */
      private double speedLimit;
      /* Used to restrict simulation speed */
      private long speedLimitLastSimtime;
      private long speedLimitLastRealtime;

      private long currentSimulationTime = 0;

      private String title = null;

      private RadioMedium currentRadioMedium = null;

      private boolean isRunning = false;

      private boolean stopSimulation = false;

      private Thread simulationThread = null;

      private static GUI myGUI = null;

      private long randomSeed = 123456;

      private boolean randomSeedGenerated = false;

      private long maxMoteStartupDelay = 1000*MILLISECOND;

      private Random randomGenerator = new Random();

      private boolean hasMillisecondObservers = false;

      private int logOutputBufferSize;

      /* Event queue */
      private EventQueue eventQueue = new EventQueue();

      /* Poll requests */
      private boolean hasPollRequests = false;
      private ArrayDeque<Runnable> pollRequests = new
ArrayDeque<Runnable>();


      private Class<? extends Mote> moteClass = null;

      public File currentConfigFile = null;

      private ArrayList<COOJAProject> currentProjects = new
ArrayList<COOJAProject>();

      private Vector<Plugin> startedPlugins = new
Vector<Plugin>();
```


IJSER © 2019
http://www.ijser.org

```
      public static Simulation mySimulation = null;

      public static ArrayList<Element> config = new
ArrayList<Element>();

      private JDesktopPane myDesktopPane;

      /*
  private SimEventCentral eventCentral = new
SimEventCentral(this);
  public SimEventCentral getEventCentral() {
    return eventCentral;
  }
       */
      /**
       * Returns all mote types in simulation.
       *
       * @return All mote types
       */
      public MoteType[] getMoteTypes() {
            MoteType[] types = new
MoteType[moteTypes.size()];
            moteTypes.toArray(types);
            return types;
      }

      /**
       * Returns mote type with given identifier.
       *
       * @param identifier
       *          Mote type identifier
       * @return Mote type or null if not found
       */
      public MoteType getMoteType(String identifier) {
            for (MoteType moteType : getMoteTypes()) {
                  if
(moteType.getIdentifier().equals(identifier)) {
                        return moteType;
                  }
            }
            return null;
      }
      /**
       * Returns simulation with with given ID.
       *
       * @param id ID
       * @return Mote or null
       * @see Mote#getID()
       */
      public Mote getMoteWithID(int id) {
            for (Mote m: motes) {
                  if (m.getID() == id) {
                        return m;
                  }
            }
```


IJSER © 2019
http://www.ijser.org

```
            return null;
      }

      public GenTestcsc()
      {
            JDesktopPane desktop = new JDesktopPane();
            myDesktopPane = desktop;
            myGUI = new GUI(desktop);
            mySimulation = new Simulation(myGUI);
      }

      /**
       * Returns number of motes in this simulation.
       *
       * @return Number of motes
       */
      public int getMotesCount() {
            return motes.size();
      }


      /**
       * Adds given mote type to simulation.
       *
       * @param newMoteType Mote type
       */
      public void addMoteType(MoteType newMoteType) {
            moteTypes.add(newMoteType);

      }

      public Mote getMote(int pos) {
            return motes.get(pos);
      }

      public void addMote(final Mote mote) {
            motes.add(mote);
      }

      /**
       * @return Max simulation speed ratio. Returns null if
no limit.
       */
      public Double getSpeedLimit() {
            if (speedLimitNone) {
                  return null;
            }
            return new Double(speedLimit);
      }

      /**
       * @return Random seed (converted to a string)
       */
      public String getRandomSeedString() {
            return Long.toString(randomSeed);
```


IJSER © 2019
http://www.ijser.org

```
	}

	/**
	 * @return Random seed
	 */
	public long getRandomSeed() {
		return randomSeed;
	}

	/* Returns the current simulation config represented
by XML elements. This
	 * config also includes the current radio medium, all
mote types and motes.
	 *
	 * @return Current simulation config
	 */
	public Collection<Element> getConfigXML() {
		ArrayList<Element> config = new
ArrayList<Element>();

		Element element;

		// Title
		element = new Element("title");
		element.setText(title);
		config.add(element);

		/* Max simulation speed */
		if (!speedLimitNone) {
			element = new Element("speedlimit");
			element.setText("" + getSpeedLimit());
			config.add(element);
		}

		// Random seed
		element = new Element("randomseed");
		if (randomSeedGenerated) {
			element.setText("generated");
		} else {

	element.setText(Long.toString(getRandomSeed()));
		}
		config.add(element);

		// Max mote startup delay
		element = new Element("motedelay_us");

	element.setText(Long.toString(maxMoteStartupDelay));
		config.add(element);

		// Radio Medium
		element = new Element("radiomedium");

	element.setText(currentRadioMedium.getClass().getName(
));
```

IJSER

```
            Collection<Element> radioMediumXML =
currentRadioMedium.getConfigXML();
            if (radioMediumXML != null) {
                  element.addContent(radioMediumXML);
            }
            config.add(element);

            /* Event central */

            /*
    element = new Element("events");
    element.addContent(eventCentral.getConfigXML());
    config.add(element);
             */

            // Mote types
            for (MoteType moteType : getMoteTypes()) {
                  element = new Element("motetype");

      element.setText(moteType.getClass().getName());

                  Collection<Element> moteTypeXML =
moteType.getConfigXML(mySimulation);
                  if (moteTypeXML != null) {
                        element.addContent(moteTypeXML);
                  }
                  config.add(element);
            }

            // Motes
            for (Mote mote : motes) {
                  element = new Element("mote");

                  Collection<Element> moteConfig =
mote.getConfigXML();
                  if (moteConfig == null) {
                        moteConfig = new
ArrayList<Element>();
                  }

                  /* Add mote type identifier */
                  Element typeIdentifier = new
Element("motetype_identifier");

      typeIdentifier.setText(mote.getType().getIdentifier())
;
                  moteConfig.add(typeIdentifier);

                  element.addContent(moteConfig);
                  config.add(element);
            }

            return config;
      }
```


IJSER © 2019
http://www.ijser.org

```
	/**
	 * @return Current desktop pane (simulator visualizer)
	 */
	public JDesktopPane getDesktopPane() {
		return myDesktopPane;
	}

	/**
	 * Sets the current simulation config depending on the
given configuration.
	 *
	 * @param configXML Simulation configuration
	 * @param visAvailable True if simulation is allowed
to show visualizers
	 * @param manualRandomSeed Simulation random seed. May
be null, in which case the configuration is used
	 * @return True if simulation was configured
successfully
	 * @throws Exception If configuration could not be
loaded
	 */
	public boolean setConfigXML(Collection<Element>
configXML,
			boolean visAvailable, Long
manualRandomSeed) throws Exception {

		// Parse elements
		for (Element element : configXML) {

			// Title
			if (element.getName().equals("title")) {
				title = element.getText();
			}

			/* Max simulation speed */
			/*
	if (element.getName().equals("speedlimit")) {
	  String text = element.getText();
	  if (text.equals("null")) {
	    setSpeedLimit(null);
	  } else {
	    setSpeedLimit(Double.parseDouble(text));
	  }
	}
			 */
			// Random seed
			if (element.getName().equals("randomseed"))
{
				long newSeed;

				if
(element.getText().equals("generated")) {
					randomSeedGenerated = true;
```


IJSER © 2019
http://www.ijser.org

```
                              newSeed = new
Random().nextLong();
                        } else {
                              newSeed =
Long.parseLong(element.getText());
                        }
                        if (manualRandomSeed != null) {
                              newSeed = manualRandomSeed;
                        }

                        mySimulation.setRandomSeed(newSeed);
                  }
                  // Max mote startup delay
                  if (element.getName().equals("motedelay"))
{
                        maxMoteStartupDelay =
Integer.parseInt(element.getText())*MILLISECOND;
                  }
                  if
(element.getName().equals("motedelay_us")) {
                        maxMoteStartupDelay =
Integer.parseInt(element.getText());
                  }

                  // Radio medium
                  if
(element.getName().equals("radiomedium")) {
                        String radioMediumClassName =
element.getText().trim();
                        Class<? extends RadioMedium>
radioMediumClass = myGUI.tryLoadClass(
                                    mySimulation,
RadioMedium.class, radioMediumClassName);

                        if (radioMediumClass != null) {
                              // Create radio medium
specified in config
                              try {
                                    currentRadioMedium =
RadioMedium.generateRadioMedium(radioMediumClass,
mySimulation);
                              } catch (Exception e) {
                                    currentRadioMedium =
null;
                              }
                        }

                        // Show configure simulation dialog
                        /*
        boolean createdOK = false;
        if (visAvailable) {
          createdOK =
CreateSimDialog.showDialog(GUI.getTopParentContainer(),
this);
        } else {
```


IJSER © 2019
http://www.ijser.org

```
          createdOK = true;
        }

        if (!createdOK) {
          throw new Exception("Load aborted by user");
        }
                        */

                       // Check if radio medium specific
config should be applied
                       /*
        if
(radioMediumClassName.equals(currentRadioMedium.getClass().g
etName())) {

currentRadioMedium.setConfigXML(element.getChildren(),
visAvailable);
        } else {
        }
                        */
                 }

                 /* Event central */
                 if (element.getName().equals("events")) {
                       // eventCentral.setConfigXML(this,
element.getChildren(), visAvailable);
                       logOutputBufferSize =
Integer.parseInt(element.getText());

                 }

                 // Mote type
                 if (element.getName().equals("motetype")) {
                       String moteTypeClassName =
element.getText().trim();
                       System.out.println("Debug:
moteTypeClassName"+moteTypeClassName);

                       /* Try to recreate simulation using a
different mote type */

                       Class<? extends MoteType>
moteTypeClass = myGUI.tryLoadClass(mySimulation,
                                  MoteType.class,
moteTypeClassName);

                       if (moteTypeClass == null) {
                            throw new
MoteType.MoteTypeCreationException("Could not load mote type
class: " + moteTypeClassName);
                       }

                       MoteType moteType =
moteTypeClass.getConstructor((Class[]) null).newInstance();
```


IJSER © 2019
http://www.ijser.org

```
                        boolean createdOK =
moteType.setConfigXML(mySimulation, element.getChildren(),
                                    visAvailable);
                        if (createdOK) {
                              addMoteType(moteType);
                        } else {
                              throw new Exception("All mote
types were not recreated");
                        }
                  }

                  /* Mote */
                  if (element.getName().equals("mote")) {

                        /* Read mote type identifier */
                        MoteType moteType = null;
                        for (Element subElement:
(Collection<Element>) element.getChildren()) {
                              if
(subElement.getName().equals("motetype_identifier")) {
                                    moteType =
getMoteType(subElement.getText());
                                    if (moteType == null) {
                                          throw new
Exception("No mote type '" + subElement.getText() + "' for
mote");
                                    }
                                    break;
                              }
                        }
                        if (moteType == null) {
                              throw new Exception("No mote
type specified for mote");
                        }

                        /* Create mote using mote type */

                        Mote mote =
moteType.generateMote(mySimulation);
                        if (mote.setConfigXML(mySimulation,
element.getChildren(), visAvailable)) {
                              if (getMoteWithID(mote.getID())
!= null) {
                              } else {
                                    addMote(mote);
                              }
                        } else {
                              throw new Exception("All motes
were not recreated");
                        }
                  }
            }
            /*
    if (currentRadioMedium != null) {
      currentRadioMedium.simulationFinishedLoading();
```


IJSER © 2019
http://www.ijser.org

```
    }
            */
           return true;
      }
      public void saveSimulationConfig(File file) {
           this.currentConfigFile = file; /* Used to
generate config relative paths */
           try {
                 this.currentConfigFile =
this.currentConfigFile.getCanonicalFile();
           } catch (IOException e) {
           }

           try {
                 // Create and write to document
                 Document doc = new
Document(extractSimulationConfig());
                 OutputStream out = new
FileOutputStream(file);

                 if (file.getName().endsWith(".gz")) {
                       out = new GZIPOutputStream(out);
                 }

                 XMLOutputter outputter = new
XMLOutputter();

      outputter.setFormat(Format.getPrettyFormat());
                 outputter.output(doc, out);
                 out.close();
                 System.out.println("Saved to file: " +
file.getAbsolutePath());

           } catch (Exception e) {
                 e.printStackTrace();
           }
      }
      public Element extractSimulationConfig() {
           // Create simulation config
           Element root = new Element("simconf");

           System.out.println("currentProjects value
"+currentProjects.toString());

           /* Store extension directories meta data */
           for (COOJAProject project: currentProjects) {
                 Element projectElement = new
Element("project");

      projectElement.addContent((project.dir).getPath().repl
aceAll("\\\\", "/"));
                 projectElement.setAttribute("EXPORT",
"discard");
                 root.addContent(projectElement);
           }
```


IJSER © 2019
http://www.ijser.org

IJSER

```
            Element simulationElement = new
Element("simulation");
            simulationElement.addContent(getConfigXML());
            root.addContent(simulationElement);

            // Create started plugins config
            Collection<Element> pluginsConfig =
getPluginsConfigXML();
            if (pluginsConfig != null) {
                  root.addContent(pluginsConfig);
            }

            return root;
      }
      public Collection<Element> getPluginsConfigXML() {
            ArrayList<Element> config = new
ArrayList<Element>();
            Element pluginElement, pluginSubElement;

            System.out.println("getPluginsConfigXML loop");
            /* Loop over all plugins */
            for (Plugin startedPlugin : startedPlugins) {
                  System.out.println("getPluginsConfigXML
loop");
                  int pluginType =
startedPlugin.getClass().getAnnotation(PluginType.class).val
ue();

                  // Ignore GUI plugins
                  if (pluginType == PluginType.COOJA_PLUGIN
                              || pluginType ==
PluginType.COOJA_STANDARD_PLUGIN) {
                        continue;
                  }

                  pluginElement = new Element("plugin");

      pluginElement.setText(startedPlugin.getClass().getName
());

                  // Create mote argument config (if mote
plugin)
                  if (pluginType == PluginType.MOTE_PLUGIN) {
                        pluginSubElement = new
Element("mote_arg");
                        Mote taggedMote = ((MotePlugin)
startedPlugin).getMote();
                        for (int moteNr = 0; moteNr <
getMotesCount(); moteNr++) {
                              if (getMote(moteNr) ==
taggedMote) {

      pluginSubElement.setText(Integer.toString(moteNr));
```


IJSER © 2019
http://www.ijser.org

```
      pluginElement.addContent(pluginSubElement);
                              break;
                        }
                  }
            }

            // Create plugin specific configuration
            Collection<Element> pluginXML =
startedPlugin.getConfigXML();
            if (pluginXML != null) {
                  pluginSubElement = new
Element("plugin_config");

      pluginSubElement.addContent(pluginXML);

      pluginElement.addContent(pluginSubElement);
            }

            // If plugin is visualizer plugin, create
visualization arguments
            if (startedPlugin.getGUI() != null) {
                  JInternalFrame pluginFrame =
startedPlugin.getGUI();

                  pluginSubElement = new
Element("width");
                  pluginSubElement.setText("" +
pluginFrame.getSize().width);

      pluginElement.addContent(pluginSubElement);

                  pluginSubElement = new Element("z");
                  pluginSubElement.setText("" +
getDesktopPane().getComponentZOrder(pluginFrame));

      pluginElement.addContent(pluginSubElement);

                  pluginSubElement = new
Element("height");
                  pluginSubElement.setText("" +
pluginFrame.getSize().height);

      pluginElement.addContent(pluginSubElement);

                  pluginSubElement = new
Element("location_x");
                  pluginSubElement.setText("" +
pluginFrame.getLocation().x);

      pluginElement.addContent(pluginSubElement);

                  pluginSubElement = new
Element("location_y");
```


IJSER © 2019
http://www.ijser.org

```
                        pluginSubElement.setText("" +
pluginFrame.getLocation().y);

      pluginElement.addContent(pluginSubElement);

                        if (pluginFrame.isIcon()) {
                              pluginSubElement = new
Element("minimized");
                              pluginSubElement.setText("" +
true);

      pluginElement.addContent(pluginSubElement);
                        }
                  }

                  config.add(pluginElement);
            }

            return config;
      }

      private Plugin startPlugin(final Class<? extends
Plugin> pluginClass,
                  final GUI argGUI, final Simulation
argSimulation, final Mote argMote, boolean activate)
                              throws
PluginConstructionException
                              {

            // Check that plugin class is registered
            /*
    if (!pluginClasses.contains(pluginClass)) {
      throw new PluginConstructionException("Tool class not
registered: " + pluginClass);
    }
             */
            // Construct plugin depending on plugin type
            int pluginType =
pluginClass.getAnnotation(PluginType.class).value();
            System.out.println("pluginType
value"+pluginType);
            Plugin plugin;

            try {
                  if (pluginType == PluginType.MOTE_PLUGIN) {
                        if (argGUI == null) {
                              throw new
PluginConstructionException("No GUI argument for mote
plugin");
                        }
                        if (argSimulation == null) {
                              throw new
PluginConstructionException("No simulation argument for mote
plugin");
                        }
```


IJSER © 2019
http://www.ijser.org

```
                        if (argMote == null) {
                              throw new
PluginConstructionException("No mote argument for mote
plugin");
                        }

                        plugin =

      pluginClass.getConstructor(new Class[] { Mote.class,
Simulation.class, GUI.class })
                                    .newInstance(argMote,
argSimulation, argGUI);

                  } else if (pluginType ==
PluginType.SIM_PLUGIN
                              || pluginType ==
PluginType.SIM_STANDARD_PLUGIN) {
                        if (argGUI == null) {
                              throw new
PluginConstructionException("No GUI argument for simulation
plugin");
                        }
                        if (argSimulation == null) {
                              throw new
PluginConstructionException("No simulation argument for
simulation plugin");
                        }


                        plugin =

      pluginClass.getConstructor(new Class[] {
Simulation.class, GUI.class})

      .newInstance(argSimulation, argGUI);

                  } else if (pluginType ==
PluginType.COOJA_PLUGIN
                              || pluginType ==
PluginType.COOJA_STANDARD_PLUGIN) {
                        if (argGUI == null) {
                              throw new
PluginConstructionException("No GUI argument for GUI
plugin");
                        }

                        plugin =

      pluginClass.getConstructor(new Class[] { GUI.class })
                                    .newInstance(argGUI);

                  } else {
                        throw new
PluginConstructionException("Bad plugin type: " +
pluginType);
```


IJSER © 2019
http://www.ijser.org

```
                  }
            } catch (PluginRequiresVisualizationException e)
{
                  PluginConstructionException ex = new
PluginConstructionException("Tool class requires
visualization: " + pluginClass.getName());
                  ex.initCause(e);
                  throw ex;
            } catch (Exception e) {
                  PluginConstructionException ex = new
PluginConstructionException("Construction error for tool of
class: " + pluginClass.getName());
                  ex.initCause(e);
                  throw ex;
            }

            if (activate) {
                  plugin.startPlugin();
            }

            // Add to active plugins list
            startedPlugins.add(plugin);
            //updateGUIComponentState();

            /*
    // Show plugin if visualizer type
    if (activate && plugin.getGUI() != null) {
      myGUI.showPlugin(plugin);
    }
             */

            return plugin;
                        }
      public class PluginConstructionException extends
Exception {
            private static final long serialVersionUID =
8004171223353676751L;
            public PluginConstructionException(String
message) {
                  super(message);
            }
      }


      public void removePlugin(final Plugin plugin, final
boolean askUser) {
            new RunnableInEDT<Boolean>() {
                  public Boolean work() {
                        /* Free resources */
                        plugin.closePlugin();
                        startedPlugins.remove(plugin);
                        //updateGUIComponentState();

                        /* Dispose visualized components */
                        if (plugin.getGUI() != null) {
```


IJSER © 2019
http://www.ijser.org

```
                        plugin.getGUI().dispose();
                    }

                    /* (OPTIONAL) Remove simulation if
all plugins are closed */
                    /*
        if (mySimulation.getSimulation() != null && askUser
&& startedPlugins.isEmpty()) {
          doRemoveSimulation(true);
        }
                     */

                    return true;
                }
            }.invokeAndWait();
        }

        public void stopAllPlugin()
        {
            for (Plugin p: startedPlugins.toArray(new
Plugin[0])) {
                removePlugin(p, false);
            }
        }

        public static abstract class RunnableInEDT<T> {
            private T val;

            /**
             * Work method to be implemented.
             *
             * @return Return value
             */
            public abstract T work();

            /**
             * Runs worker method in event dispatcher
thread.
             *
             * @see #work()
             * @return Worker method return value
             */
            public T invokeAndWait() {
                if(java.awt.EventQueue.isDispatchThread())
{
                    return RunnableInEDT.this.work();
                }

                try {
                    java.awt.EventQueue.invokeAndWait(new
Runnable() {
                        public void run() {
                            val =
RunnableInEDT.this.work();
                        }
```

IJSER

IJSER © 2019
http://www.ijser.org

```
                });
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }

            return val;
        }
    }

    public static void main(String args[])
    {
        String fullFile = new String() ;
        String testCaseName = new String();
        boolean SimControl = false;
        boolean TimeLine = false;
        //GenTestcsc test = new GenTestcsc();
        try{
            fullFile = new
String(Files.readAllBytes(Paths.get("GenTest.txt")));
        } catch (Exception e) { e.toString();}
        Pattern pat = Pattern.compile("[\\(\\)]");
        String strs[] = pat.split(fullFile);
        for(int i=0; i<strs.length;i++)
        {
            System.out.println("strs[ "+i+"]"+strs[i]);
            if(strs[i].length() < 10) continue;
            Pattern pat1 = Pattern.compile("[\\{\\}]");
            String strs1[] =
pat1.split(strs[i].trim());
            for (int j=0; j<strs1.length;j++)
            {
                System.out.println("Next token
:"+strs1[j].trim());
                Pattern pat2 =
Pattern.compile("[\\,]");
                if(strs1[j].contains("testcasename"))
                {
                    String strs2[] =
pat2.split(strs1[j].trim());
                    System.out.println("Test case
name is "+strs2[1]);
                    testCaseName=strs2[1].trim();
                }
                if(strs1[j].contains("title"))
                {
                    String strs2[] =
pat2.split(strs1[j].trim());
                    Element temp = new
Element("title");
                    temp.setText(strs2[1]);
                    config.add(temp);
                }
                if(strs1[j].contains("radiomedium"))
```


IJSER © 2019
http://www.ijser.org

```
                        {
                              String strs2[] =
pat2.split(strs1[j].trim());
                              Element temp1 = new
Element("radiomedium");
                              temp1.setText(strs2[1]);
                              config.add(temp1);
                        }
                        if(strs1[j].contains("motetype") &&
!strs1[j].contains("mote1"))
                        {
                              String strs2[] =
pat2.split(strs1[j].trim());
                              for(int k=0;k<strs2.length;k++)
                                    System.out.println("Strs2
"+strs2[k]);
                              Element temp2 = new
Element("motetype");
                              temp2.setText(strs2[1].trim());
                              Element sourceElem = new
Element("source");

      sourceElem.setText(strs2[3].trim());
                              Element identifierElem = new
Element("identifier");

      identifierElem.setText(strs2[5].trim());
                              temp2.addContent(sourceElem);

      temp2.addContent(identifierElem);
                              config.add(temp2);
                        }
                        if(strs1[j].contains("mote1"))
                        {

                              String strs2[] =
pat2.split(strs1[j].trim());
                              Element temp3 = new
Element("mote");
                              temp3.setText(strs2[1].trim());
                              Element temp4 = new
Element("motetype_identifier");
                              temp4.setText(strs2[3].trim());
                              temp3.addContent(temp4);
                              config.add(temp3);
                        }
                        if(strs1[j].contains("SimControl"))
                        {
                              SimControl = true;
                        }
                        if(strs1[j].contains("TimeLine"))
                        {
                              TimeLine = true;
                        }
                  }
```


IJSER © 2019
http://www.ijser.org

```
                  System.out.println("config to
string"+config.toString());

                  Long manualRandomSeed = new Long(1);
                  try{
                        GenTestcsc test = new GenTestcsc();
                        try{


      myGUI.setVisualizedInFrame(false);
                              Class pluginClass =
Class.forName("se.sics.cooja.plugins.SimControl");
                              Class pluginClass1 =
Class.forName("se.sics.cooja.plugins.TimeLine");

      //test.startPlugin(pluginClass,myGUI,mySimulation,null
,true);

      //test.startPlugin(pluginClass1,myGUI,mySimulation,nul
l,true);
                              if(SimControl)
                              {

      //test.startPlugin(pluginClass,myGUI,mySimulation,null
,true);
                              }
                              if(TimeLine)
                              {

      //test.startPlugin(pluginClass1,myGUI,mySimulation,nul
l,true);
                              }
                        } catch( Exception e)
                        {
                              System.out.println("Exception
while starting the plugin: " + e);
                              e.printStackTrace();
                        }
                        File file = new File(testCaseName);
                        file.createNewFile();

      test.setConfigXML(config,false,manualRandomSeed);
                        test.saveSimulationConfig(file);
                        System.out.println("After stop
simulation");
                        mySimulation.stopSimulation();
                        myGUI.doRemoveSimulation(false);
                        test.stopAllPlugin();
                        config = new ArrayList<Element>();

                  } catch (Exception e)
                  {
                        System.out.println("Exception while
saving simulation config: " + e);
                        e.printStackTrace();
```


IJSER © 2019
http://www.ijser.org

```
                }
            }
        }
    }
```

IJSER

IJSER © 2019
http://www.ijser.org

Appendix G: Candidate's Biography

Name	Abhinandan H Patil	
Education	Secondary School 83.04%	Belagavi, India
	Pre University 80%	Belagavi, India
	B.E (Electronics and Communication) 75.8%	GIT, Belagavi, India
	M. Tech (Computer Science and Engineering) CGPA 7.85	VTU, Belgavi, India
Experience	Industrial exposure	Infosys : 0.5 Years Motorola: 8. 4 Years Kshema Tech : 1.3 Years

IJSER

Abhinandan H Patil is a research student at BITS Pilani, Goa campus since Jan 2014. Before joining BITS Pilani, Goa he was working in wireless software industry mainly on wireless simulator for telecom network. His research interests include, wireless networks, simulators for wireless networks, software engineering, software testing, combinatorial testing. He is now exploring artificial intelligence and machine learning, data analytics and cloud computing.

IJSER © 2019
http://www.ijser.org

Appendix H: Publications of The Candidate

Publications from Thesis

[1] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan,"Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques", International Journal of Education and Management Engineering(IJEME),2016,Vol.6, No.5, pp.32-39, 2016.DOI: 10.5815/ijeme.2016.05.04

[2] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan,"Regression Test Suite Execution Time Analysis using Statistical Techniques", International Journal of Education and Management Engineering(IJEME),2016, Vol.6, No.3, pp.33-41, 2016.DOI: 10.5815/ijeme.2016.03.04

[3] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan,"Test Suite Design Methodology Using Combinatorial Approach for Internet of Things Operating Systems," Journal of Software Engineering and Applications,2015, 8, 303-312. doi: 10.4236/jsea.2015.87031

[4] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan,"Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach," ACM SIGSOFT SEN, 2015, Volume 40 Issue 2, pp 1-3,doi:10.1145/2735399.2735413

[5] Abhinandan H. Patil, Preeti Satish, Neena Goveas and Krishnan Rangarajan,"Integrated test environment for combinatorial testing,"Advance Computing Conference (IACC), 2015 IEEE International, 2015, doi: 10.1109/IADCC.2015.7154802

[6] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan, “Generating Effective Test Suite for Multiparameter Software using ACTS Tool and its Verification using Code Coverage Tools”, 2018, IJSER, Volume 9, Issue 8.

[7] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan, "Design and Implementation of Contiki and Cooja Regression Test Suites by Using Combinatorial Testing", Communicated.

IJSER

IJSER © 2019
http://www.ijser.org

[8] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan, " ACTS-RT: Advanced Combinatorial Testing for Software based Regression Testing and its application to IoT Operating System Contiki”, Communicated

[9] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan, “CT-RTS: Combinatorial Testing based Regression Test Suite: Functional Test Case Generator for Contiki and Cooja”, Communicated.

Other Publications

[1] “CodeCover: enhancement of CodeCover,” ACM SIGSOFT Software Engineering Notes 02/2014; 39(1):1-4., DOI:10.1145/2557833.2557850

[2] “CodeCover: A Code Coverage Tool for Java Projects”, Elsevier, ERCICA; 03/2013

IJSER

IJSER © 2019
http://www.ijser.org

Appendix I: Supervisors Biodata

Neena Goveas is with the Department of Computer Science at BITS Pilani K K Birla Goa campus.
For her PhD thesis, she worked on "Mean field approaches to thermodynamic properties of magnetic systems" at IIT Bombay, advisor Prof. G. Mukhopadhyay. She worked on INDO-US sponsored project "Development and characterization of materials suitable for magneto-optic Devices" at A. C. R. E., I. I. T. Bombay. She continued her research as a DST-Young Scientist in a Project entitled "Study of low dimensional magnetic systems" at IIT Guwahati. Other projects she was associated as a PI or Co-PI are “Development of Remotely Configurable Arbitrary Ramp Generator for FMCW Reflectometry, BRNS” and “Implementation of Wireless Sensor Network for Process Monitoring of GAIL’s Pipeline, GAIL India Ltd”.

Her main theme of research work is to study complex systems. Using various mean field and computational approaches to understand their properties. Recent research work is on Network Science and its applications to transport, social and computer networks; modeling of Cyber Physical Systems and Wireless Sensor Networks; Construction of test suites for large software systems.

IJSER

IJSER © 2019
http://www.ijser.org

Appendix J: Co-Supervisors Biodata

R. Krishnan, Ph. D

Contact Details:

Mobile: 9844264071; **E-mail:** krishnanr1234@gmail.com

Educational Qualification

- Ph.D (Computer Science) with specialisation in computer vision, from University of Central Florida, Orlando, USA 1987-90; GPA 4 on a 4 point scale
- M.Tech (Computer Science) from IIT (Indian Institute of Technology) , Delhi, India 1983-85; GPA 9.8 on a 10 point scale
- B.E(Hons) Mechanical Engineering from Regional Engineering College, Tiruchirapalli, India 1978-83; scored 83%, ranked 6th in the University of Madras, India.

IJSER

Work Experience:Academics

<u>Current Position:</u> **Professor, Department of Computer Science and Engineering @ Dayananda Sagar College of Engineering, Bangalore**

Teaching Interests: Software Testing, Object Oriented Modelling & Design, Software Architecture and AI.

Research: Research interests include Object tracking in Computer Vision , Software engineering topics like Combinatorial Testing, Usability, software architecture. Guiding six research scholars in the areas of computer vision and software testing.

Publications: International : 45, National : 7, citations: 693, h-index: 9 , i-10 index: 8

- Was in the program committee of IWCT (International Workshop on Combinatorial Testing) 2015 , 2016 held in conjunction with IEEE ICST (International conference on Software Testing) 2015,2016.
- Was a visiting research scholar for a month (June 2011 – July 2011) in computer vision lab, University of Central Florida, Orlando, USA. This lab is

IJSER © 2019
http://www.ijser.org

headed by Dr Mubarak Shah, a leading researcher in video processing, who was also the advisor for my Ph D work on Motion trajectories. I also attended the IEEE conference CVPR 2011(Computer Vision and Pattern recognition 2011), Colarado, USA in this period.

Sponsored projects handled as PI:

SL No.	Project Title	Sponsoring agency	Date of Sanction	Grant Amount	Status
1	Multi Object tracking in the presence of occlusion in aerial image sequences	ER&IPR, DRDO	11/2/2017	Rs 22.07 Lakhs	Ongoing
2	Tracking multiple objects in Aerial image sequence from an Unmanned Air Vehicle	ER&IPR, DRDO	18/8/2011	Rs 21.20 Lakhs	Completed

Industry interaction:

- Conducted training on Combinatorial testing to industry (LG, Testing workshop)
- Delivered training in Motorola on Software Architecture & Design Patterns
- Interacting with CISCO for establishing center of excellence in networking
- Interacting with a start-up in setting up of AI Lab
- Engaged as an external software engineering expert in six sigma improvement initiatives with the Indian operations of a leading consumer electronics major.

IJSER

Professor, Department of Computer Science and Engineering , CMRIT, Bangalore(July 2015 – Nov 2016).

- Taught courses on Object Oriented Modelling & Design, Software Engineering for UG and Cloud Computing, AI&Agent Technology, Machine Learning Techniques for PG .
- Organized five day IEEE Workshop on Applications and Research Directions in Big Data, Dec 9 -13, 2015, CMRIT Bangalore.
- POC for MOU with Delphi Automotive systems and conducted a half day workshop on Combinatorial Testing in Delphi center, Nov 2015.
- Publication chair for IEEE CCEM 2016.

IJSER © 2019
http://www.ijser.org

Professor, Department of Computer Science and Engineering , SSN Institutions, Chennai (Jan 2009 – Oct 2009 :)

Jan 2009 – Mar 2009: Worked in SSN School of Advanced software engineering, Chennai

April 2009 – Oct 2009: Worked in the Department of Computer Science & Engineering, SSN College of Engineering, Chennai. Taught the course on Software Quality Assurance for the M.E Computer Science & Engineering students and handled the case tools lab for the B.E Computer Science & Engineering final year students.

Work Experience:Software Engineering

Motorola India: (April 1996 – Oct 2008)

Designation: Principal Staff Engineer

Technical Lead & manager for a small **applied research** team in the **software engineering tools & technology** area. Team's charter was to identify, evaluate, pilot & induct software engineering tools, methods/practices that help to improve productivity/quality in software and champion software engineering initiatives in the organization. The role involved working closely with the project teams in driving the software engineering roadmap for the organization.

Software engineering areas worked on:

- **Software Testing & Automation**
- Was the site level champion for test interest group and represented the site in the Corporate level Test Process Improvement group.
- Initiated development of an in-house tool to support testcase generation based on OATS (Orthogonal array Based Testing Strategy) and championed its use successfully in the organization resulting in huge effort savings in testing involving combinations.
- Evaluated unit test tools and made recommendation to the organization. Championed the adoption of the recommended tool. Statistical skills were demonstrated in data analysis in this project and was certified as a Six Sigma green belt.

IJSER © 2019
http://www.ijser.org

- Led a team of two members in the enhancement and support of a remote testing tool VMD (Virtual Mobile Device) for remote testing mobile devices. This involved understanding the issues and gaps in the tool based on usage in project teams and working with the vendor to get them addressed.
- Was the site level champion for security testing tools like Mu, Codenomicon.
- Identified and Piloted the use of AspectC++ tool for automatic logging & tracing. Interacted with the university professor and got few show stoppers in this tool addressed.
- Managed a small team working on Model driven Engineering and championed the use of the technology
- Created software testing course material covering test related concepts, process and tools for training new entrants into the organization. This course was developed with contributions from the practioners in the project teams.
- **Software Security**

Was the site level champion for this corporate initiative, working closely with the corporate security champion. Key contributions include:

- Collaborating with external security experts in defining the security rules in the coding standards and getting them supported in the static analysis tool widely used in Motorola.
- Revising the existing software process to absorb this practice.
- Interacted with the project teams to help them understand the initiative and the related practices. Was also the Lead trainer at the site level for the secure programming course: an awareness course mandated for all the developers.
- Managed the team that supported the related tools
- **Product Quality**

This initiative was designed to bring focus on product quality aspects like performance, availability, usability etc during the development cycle. Took a lead role in formulating a framework for improving product quality, created the related process assets & training material, piloted and inducted the practices like attribute specification, architecture evaluation, usability inspections and usability feedback survey using SUMI (Software Usability Measurement Inventory) method. Partnered with the quality department and product quality champions from Operations, in rolling out this initiative. This

IJSER © 2019
http://www.ijser.org

was recognised as an "Emerging Best Practice" in Motorola Software Engineering Symposium-2000, a paper describing this work was published in ACM Software Engineering Notes, July-2001 pp:77-82 ; Also presented a tutorial on this topic in SEPG International conference on software engineering – 1999.

- **Software Reuse**
- Championed reuse in Motorola India for couple of years. Identified key domains and formed reuse champions team and domain teams. It was essentially an opportunistic reuse program, centered around an in-house repository tool. Educated the project teams on reuse maturity models and the importance of moving to the highest maturity (the twin lifecycle model). Contributed to a more recent corporate level asset based reuse program, in creating and rolling out an asset evaluation scheme.

- **Software Architecture & Design**
- Created and delivered a tutorial on Software architecture as part of Motorola Technology Seminar series and in SEPG International conference on software engineering-2001
- Identified and Piloted architecture evaluation methods like SAAM(Software architecture analysis method), ATAM(Architecture Tradeoff Analysis Method) in projects.
- Did a survey of published work in software design for multicore and synthesized a set of design guidelines. Delivered a technical talk on this topic in an internal technical forum .
- Conducted internal courses on Software Design & Design Patterns.
  - **Process Mapping**
- Participated in the process mapping exercise undertaken at Motorola India, in collaboration with corporate software engineering research labs.
  - Helped apply system dynamics modelling concept to the recruitment problem.
- **Other work in the Tools Area**
- Coordinated the evaluation and induction of various software engineering tools like Purify, TestExpert, Xrunner, Source Insight etc.
- Customer interface and people manager for a 4 member tool support team

Tata Consultancy Services, Chennai :May 1992 – March1996

IJSER © 2019
http://www.ijser.org

Designation: Manager

January 1994 - March 1996 @ TCS Chennai

- Software Implementation of MPEG-1 audio encoder using 'C' under windows. The challenge was in getting a realtime implementation to match sound capture rate.
- Modeled software project management using system dynamics concepts. Developed a simulation engine for system dynamic models and a front-end for software project management problem. The tool was used for simulating few scenarios
- Did a study to consolidate the problems faced by the language processing tools team and this led to the understanding of issues to be addressed by R & D.
- Guided few student projects

November 1992 - December 1993 @AT&T Bell Labs, Merrimack Valley, USA (Worked as a consultant from TCS)

- This was a contract assignment through TCS, Chennai. A member of the software conveyor belt team, an international team formed to bring in systematic reuse in their Transmission Business Unit. Participated in the design and development of a toolset around a MIL (Module Interconnection Language). Also worked on a demo prototype using the MIL. This work involved hooking different communication mechanisms like shared memory to the MIL framework. This work was done using 'C' and Meta tool an application generator. Also participated in brainstorming sessions, discussions and reviews relating to multiuse design.

May 1992 - October 1992 @ TCS Chennai

- Was attached to the training department (Visa waiting period)

CMC R&D Lab, Secunderabad, India : January 1985 - July 1987

Designation: Engineer

- A member of product development team. Was involved in the design and development of a number of interfaces and utilities to ADMIN, a network model database package.

Work Experience : Computer Vision Research

IJSER

IJSER © 2019
http://www.ijser.org

Centre for Artificial Intelligence & Robotics, Bangalore : October 1990 - April 1992

Designation: Scientist “C”

- The work included establishing a research facility in Computer Vision along with a senior colleague and guiding a team on various research topics in Computer Vision like stereo vision and shape based object recognition.

University of Central Florida, Orlando, USA : August 1987 - July 1990

- Was a research assistant with Prof. Mubarak Shah in the Computer Science Department. In addition to doing my thesis work in Computer Vision, helped few undergraduate students in their research projects. The thesis was on “Motion Trajectories”, addressed the sub problems on model based object recognition through Motion such as trajectory generation, segmentation and matching.

PUBLICATIONS

IJSER

Software Engineering Journals

1.Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques", Abhinandan H. Patil a*, Neena Goveas a, Krishnan Rangarajan, I.J. Education and Management Engineering, 2016, 5, 32-39 Published Online September 2016 in MECS.

2."A Preliminary Survey on Combintorial Test Design Modeling Methods" , Preeti S, Krishnan Rangarajan,, IJSER Volume 7, Issue 7, July 2016.

3.Abhinandan H. Patil, Neena Goveas, Krishnan Rangarajan “Test Suite Design Methodology Using Combinatorial Approach for Internet of Things Operating Systems”, Journal of Software Engineering and Applications, 2015, 8, 303-312, Published Online July 2015 in SciRes.

4.Abhinandan H. Patil, Neena Goveas, Krishnan Rangarajan , " Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach ", ACM Software Engineering Notes, Volume 40 Issue 2, March 2015.

5. Shubha Bhat, Vindhya Malagi, Krishnan Rangarajan, Ramesh Babu D.R, “Computer Vision Guided based guidance in UAVs: Software Engineering

IJSER © 2019
http://www.ijser.org

challenges", ACM Software Engineering Notes, Volume 35, Number 6, July 2010
6. R Krishnan, Margaret Nadworny, Nishil Bharill, "Static Analysis tools for security checking in code at Motorola, ACM SIGAda Ada Letters, Volume XXVIII Issue 1, April 2008, pages 76-82.
7. R Krishnan, S Murali Krishna, Nishil Bharil, "Code Quality Tools: Learning from our Experience", ACM Software Engineering Notes, Volume 32, Number 4, July 2007
8. R Krishnan, S Murali Krishna, P Siva Nandhan, "Combinatorial Testing: Learnings from our Experience", ACM Software Engineering Notes, Volume32, Number 3, May 2007
9. Krishnan Rangarajan et.al, Product Quality Framework, published in ACM software engineering notes, Volume 26, Number 4, July 2001.

Software Engineering Conferences:

1. "Broken Kannada Character Recognition- a Neural Network based approach", Sandhya.N, Krishnan. R, D.R.Ramesh Babu , IEEE-ICEEOT-2016 , March 3 - 6, DMI College of Engineering, Chennai.
2. Abhinandan H Patil, Preeti Satish, Krishnan R, "Integrated test environment for combinatorial testing", IEEE IACC 2015, june 2015, Bangalore, India.
3. Preethi Satish, Arinjita Paul, Krishnan Rangarajan, "Extracting the Combinatorial Test Parameters and Values from UML Sequence Diagrams", icstw, 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, Cleveland, Ohio USA, March 31, 2014.
4.Preethi Satish, Sheeba K, Krishnan Rangarajan, "Deriving Combinatorial Test Design Model from UML Activity Diagram", icstw, pp 331-337, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, Luxemberg, 2013.
5. R Krishnan, Nishil Bharil, Margaret Nadworny, "Static Analysis Tools for checking Security in Code at Motorola" in Static Analysis Summit - II conference, Nov 8-9, 20 07, Fairfax, Virgina, USA
6. R Krishnan, Margaret Nadworny, "Moving Software from Expense to Asset" presented in IEEE-Compsac, September 2006, Chicago
7. Murali Krishna, R Krishnan, "Cost of Quality Reduction through Unit Test Automation, International Conference on Software Testing", January 20 – 22, 2005, Taj Residency, Bangalore, India.

IJSER

Software Engineering (On file in Motorola Symposiums):

1. Software Systems and Simulation, SSS 2006, Tools fair: Poster on security rules support in Klocwork

2. Software Systems and Simulation, SSS 2005, "Orthogonal Array Test Strategy"

3. Software Systems and Simulation, SSS 2005, Tools fair : Moto_Oatsgen (an internal tool supporting Orthogonal Array Testing).

4. Motorola European Test Symposium – 2005, presentation on "Orthogonal Array Test Strategy"

5. Software Systems and Simulation, SSS 2004 "Viewpoints Methodology: A structured, integrative and stakeholder-driven approach to requirements elicitation"

6. Software Systems and Simulation, SSS 2003 "A systematic approach to improve usability of a product"

7. Software Systems and Simulation, SSS 2003 "Aspect Oriented Programming"

8. Software Systems and Simulation, SSS 2003 "Architecture Evaluation Using ATAM"

9. Software Engineering Symposium, SES 2001 "Architecture Technology Map"

10. Motorola System Engineering Symposium - MSS 2000 "Product Quality Framework: A vehicle for focussing on High Availability and other Design Goals"
11. Software Engineering Symposium, SES 2000 "Domain Analysis of Protocol test environment"
12. Asia Pacific Software Engineering Symposium- APSES 1999 "Testing for non-functional attributes"
13. Asia Pacific Software Engineering Symposium- APSES 1999 "Usability Analysis with SUMI Method"
14. Asia Pacific Software Engineering Symposium- APSES 1999 "Product Quality Framework"
15. Asia Pacific Software Engineering Symposium- APSES 1997 "Reuse Economic Model"

Software Engineering Tutorials presented in Conferences, Technical Forums

1. Conducted sessions on "Combinatorial Testing" and "Security practices in the Software Lifecycle" in the workshop on Methodologies for Effective

IJSER © 2019
http://www.ijser.org

Software Testing, conducted by Innovate-IT in Bangalore, March 11-12, 2011. http://www.innovate-it.in/workshop_software_testing.html

2. Invited talk on “Software Quality” in VITCON-2007, organized by Vemana Institute of Technology, April 28th, 2007

3. Invited colloquim talk on Usability organized by ISQT, Bangalore June 2007

4. Was a trained trainer for the Secure Programming Course in Motorola (2005-07). Initiated trainers for this course in couple of other units in Motorola

5. Created & delivered a talk on Software Architecture, Feb 2002 which is available as an online training material with Motorola University

6. Invited lecture on Software Reuse in Leadership meet of Honeywell, June 9, 2001

7. Invited talk on Product Quality in Bangalore SPIN, May 24, 2001

8. Tutorial on Architecture in international conference on software engineering, SEPG 2001, Bangalore, India

9. Tutorial on Product Quality in international conference on software engineering, SEPG-99, Bangalore, India

10. Tutorial on reuse. In conference on software engineering, CONSEG-97, Madras, India

Computer Vision Journals:

1. Multi-object Tracking in Aerial Image Sequences using Aerial Tracking Learning and Detection Algorithm,Vindhya P. Malagi*, Ramesh Babu D.R., and Krishnan Rangarajan ,Defence science Journal, Vol. 66, No. 2, march 2016, pp 122-129

2."Anjana B.H, Rashmi S, R, Krishnan, ""A Survey on Context Driven activity Recognition and Analysis in Wide Area Surveillance", in International Journal of Ethics in Engineering & management education [ISSN: 2348-4748], Volume 2, Issue 5, May 2015, pp 36-42.

2. An algorithm to estimate scale weights of complex wavelets for Effective Feature Extraction in Aerial Images, Shubha Bhat, Ramesh Babu D.R, Krishnan Rangarajan, Ramakrishnan K.A, Defence science Journal, Vol 64, No 6, 2014.

3. Sandhya.N, R Krishnan and Ramesh D R Babu. : Feature Based Kannada Character Classification Method of Kannada Character Recognition. IJSER Volume 5, Issue 2, Feb 2014.

4. Sandhya.N, R Krishnan and Ramesh D R Babu. : A Language Independent Characterization of Document Image Noise in Historical Scripts. International Journal of Computer Applications 50(9):11-18, July 2012.

IJSER © 2019
http://www.ijser.org

IJSER

5.Ashish Sethi, Hemanth S,Kuldeep Kumar, Bhaskara Rao N, Krishnan R: SignPro-An Application Suite for Deaf and Dumb, IJCSET, May 2012, Vol 2, Issue 5,1203-1206, ISSN: 2231-0711.

6.L.N. Mohankumar, R. Kishnan, and V.P. Malagi :An Efficient Approach for Identification and Extraction of Moving Objects in Video Sequences Using Morphological Dilation, International Journal of Research and Reviews in Computer Science (IJRRCS), Vol. 2, No. 5, October 2011, ISSN: 2079-2557 © Science Academy Publisher, United Kingdom.

7. Shubha Bhat, Ramesh Babu D.R, Krishnan Rangarajan, Ramakrishnan K.A, "An algorithm to estimate scale weights of complex wavelets for Effective Feature Extraction in Aerial Images", Defence science Journal, Vol 64, No 6, 2014.

8.Krishnan Rangarajan, A. G. Seethalakshmy, Jharna Majumdar: "Computation and use of planar face normal", Pattern Recognition Letters 14(10): 809-816 (1993)

9. Mubarak Shah, Krishnan Rangarajan, Pins Sing Tsai, "Motion Trajectories", published in "IEEE transaction on systems, man and cybernatics". Also was accepted for presentation in "International Conference on pattern recognition", Hague, Holland, August 31- September 3, 1992

10. K.Rangarajan, William Allen and M.Shah, "Matching Motion Trajectories using scale space", Journal of pattern recognition Vol 26, 004, pp 595-610,1993.Also was accepted for presentation in "International Conference on Pattern Recognition", Hague, Holland, August 31- September 3, 1992.

11. K. Gould, K. Rangarajan and M.Shah, "Trajectory Primal Sketch" appeared in the book "Advances in Image Analysis", edited by Mahdeviah and Gonzalez, published by the optical engineering press.

12. K. Rangarajan and M. Shah, "Interpretation of Motion Trajectories using Focus of Expansion" appeared in the Journal "IEEE Transactions on Pattern Analysis and Machine Intelligence", Vol 14, No 12, December 1992, pp 1205-1210

13. K. Rangarajan and M.Shah, "Establishing Motion Correspondence" was presented in "IEEE Computer Society Conference on Computer Vision and Pattern Recognition" June 1991 at Hawaii. It also appeared in the Journal "CVGIP: Image Understanding", Vol 54, pp 56-73 (July 1991).

14. K. Rangarajan, M.Shah and D. Van Brackle, "Optimal Corner Detector", was presented in Second International Conference on Computer Vision, December

IJSER © 2019
http://www.ijser.org

1998 at Tampa, Florida. This paper also appeared in the Journal "Computer Vision, Graphics and Image Processing", Vol 48, pp 230-245 Nov 1989.

Computer Vision Conferences:

1. Sandhya.N, Krishnan. R, D.R.Ramesh Babu,"A novel local enhancement technique for rebuilding broken characters in degraded Kannada scripts ", IEEE IACC 2015, june 2015, Bangalore, India.

2. Sandhya N, Krishnan R, Ramesh Babu D.R, "Handwritten Kannada Character Recognition using Zonal Features and Multi-class SVM", ICPRMSP , Jan 9-10 2015, Annamalai University, India.

3. Sandhya.N, Krishnan. R, D.R.Ramesh Babu, Pianka Das, “A Comprehensive pre-processing approach for digital preservation of documents”, Proceedings of International Conference on Emerging Research in Computing, Information, Communication and Applications, ERCICA-2014, August 01-02, Bangalore, India.

4. Shubha Bhat, Ramesh Babu D.R, Krishnan Rangarajan, Ramakrishnan K.A, “Evaluation of feature descriptors to recover camera parameters for navigation of unmanned air vehicles”, Proceedings of International Conference on Emerging Research in Computing, Information, Communication and Applications, ERCICA-2014, August 01-02, Bangalore, India.

5.Vindhya Malagi, Vinutha Gayathri, Krishnan Rangarajan, Ramesh Babu D.R, “Enhancing COCOA framework for tracking moving objects in the presence of occlusion in Aerial Image Sequences”, ICMCCA 2012, Dec 13-15, 2012, Bangalore, India.

6. Jharna Majumdar, Adil Hamid and Krishnan Rangarajan "CAD Model Based System for Visual Inspection", was accepted for presentation in Second International Conference on Automation, Robotics and Computer Vision, September 15-18, 1992, Singapore.

7. Krishnan Rangarajan, Seethalakshmy and Jharna Majumdar, "Computation and use of Planar Face Normals", was accepted for presentation in Second International Conference on Automation, Robotics and Computer Vision, September 15-18, 1992, Singapore.

8. Monisha Dhar, Krishnan Rangarajan and Jharna Majumdar, "Edge and Region Based Stereo", was presented in IPA Conference cum workshop on AI applications in physical sciences, January 15-17, 1992 at BARC, Bombay. Also appeared in "Asia-Pacific Engineering Journal (APEJ)", (Part A), Vol 2, No 2, 1992 pp 217- 231.

IJSER

9. Seethalakshmy, Krishnan Rangarajan and Jharna Majumdar, "Part Identification for Robotic Applications", was presented in IPA Conference cum workshop on AI applications in physical sciences, January 15-17, 1992 at BARC, Bombay.

10. Mubarak Shah, Krishnan Rangarajan, Pins Sing Tsai, "Motion Trajectories", "International Conference on pattern recognition", Hague, Holland, August 31- September 3, 1992

11. K.Rangarajan, William Allen and M.Shah, "Matching Motion Trajectories using scale space", "International Conference on Pattern Recognition", Hague, Holland, August 31- September 3, 1992.

12. K. Rangarajan and M.Shah, "Establishing Motion Correspondence" was presented in "IEEE Computer Society Conference on Computer Vision and Pattern Recognition" June 1991 at Hawaii.

13. K. Rangarajan, M.Shah and D. Van Brackle, "Optimal Corner Detector", was presented in Second International Conference on Computer Vision, December 1988 at Tampa, Florida.

IJSER

IJSER © 2019
http://www.ijser.org

References

[1] K. Ashton, "Internet of things," *RFIDJournal,* 2009.

[2] Dunkels, J. P. Vasseur and A., Interconnecting Smart Objects with IP The Next Internet, Morgan Kauffman Publishers, 2010.

[3] H. Chaouchi, The Internet of Things, John Wiley and Sons, 2010.

[4] D. A. P and V. J., "IP for Smart Objects," 8 10 2017. [Online]. Available: http://www.ipso-alliance.org.

[5] B. N, J. M, D. A and V. T, "The design and implementation of an ip-based sensor network for intrusion monitoring," 2006.

[6] B. T. Myers, G. J. and T. M. Thomas, "The Art of Software Testing," 2010.

[7] Contiki, "Contiki Web Site," 2017. [Online]. Available: http://www.contiki-os.org/.

[8] A. P. Mathur, Foundations of Software Testing, Pearson, 2013.

[9] P. C.Jorgensen, Software Testing A Craftsman's Approach, CRC Press, 2013.

[10] "TinyOS website," 8 10 2017. [Online]. Available: http://www.tinyos.org.

[11] "TinyOS Working Group," 8 10 2017. [Online]. Available: http://www.cs.utah.edu/ regehr/tinyos-tools-wg/.

[12] "RIOT website," 8 10 2017. [Online]. Available: http://www.riot-os.org/.

[13] Y. Lee, D. Kuhn and R. N. Kacker, "Practical Combinatorial Testing Manual," 2017.

[14] C. Nie, *Practical Combinatorial Testing,* ACM Survey, 2014.

[15] Y. L. L. Shikh, G. Ghandehari and M. N. Bourazjany, "An input space modeling methodology for combinatorial testing," in *International Workshop on Combinatorial Testing*, 2013.

[16] P. B. P. Ammann, "Abstracting Formal Specifications to Generate Software Tests via Model Checking," in *Proc. 18th Digital Avionics Systems Conference*, 1999.

IJSER © 2019
http://www.ijser.org

[17] A. G. P. V. Paolo Arcaini, "Validation of models and tests for constrained combinatorial interaction testing," in *IEEE International Conference on Software Testing, Verification, and Validation Workshops*, 2014.

[18] Johansen and M. F., *Testing Product Lines of Industrial Size: Advacements in Combinatorial Interaction Testing,* University of Oslo, PhD Thesis, 2013.

[19] "ACTS tool website," 2017. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/index.html.

[20] "CCM Tool website," 2017. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/index.html.

[21] M. N.Borazjany, "Combinatorial testing of acts: A case study," in *Fifth International Conference on Software Testing, Verification and Validation*, 2012.

[22] Y. Lee, D. Kuhn and R. Kacker, "Estimating fault detection effectiveness," in *IEEE International Conference on Software Testing, Verification, and Validation Workshops*, 2014.

[23] L. Yu., D. Kuhn and R. Kacker, "Combinatorial coverage measurement concepts and applications," in *International Workshop on Combinatorial Testing*, 2013.

[24] OpenClover, "Open Clover," 2017. [Online]. Available: http://openclover.org.

[25] "CodeCover Web site," 2017. [Online]. Available: http://codecover.org.

[26] M. J. Harrold, J. A. Jones, D. L. T. Li, A. Orso, M. Pennings, S. Sinha, S. A. Spoon and A. Gujarathi, "Regression test selection for java software," in *Proceedings of the 16th ACM SIGPLAN conference on Object Oriented Programming, Systems, Languages, and Applications*, 2001.

[27] L. Yu., D. Kuhn and R. N.Kacker, Introduction to Combinatorial Testing, A Chapman and Hall Books, 2013.

[28] NIST, "NIST Website," 2017. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/index.html.

[29] P. J. a. H. Wallinder, "A Test Tool Framework for an Integrated Test Environment in the Telecom Domain," 2017.

IJSER

IJSER © 2019
http://www.ijser.org

[30] Y. Lee, L. Shikh, G. Ghandehari and M. N. Bourazjany, "Applying combinatorial testing to the siemens suite," in *International Workshop on Combinatorial Testing*, 2013.

[31] CTT, "College Time Table," 05 Sept 2018. [Online]. Available: https://sourceforge.net/projects/collegetimetable/.

[32] A. H. Patil, N. Goveas and K. Rangarajan, "Clover Logs of Execution," 10 2017. [Online]. Available: https://drive.google.com/drive/folders/0B2vHzPHgs0nVZWxtSE5sVVdGUmc.

[33] "Contiki supported hardware platforms," 2017. [Online]. Available: http://www.contiki-os.org/hardware.html.

[34] A. H. Patil, N. Goveas and K. Rangarajan, "Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach," *ACM SIGSOFT SEN,* 2015.

[35] J. T. Pro, "J Test Pro," 10 2017. [Online]. Available: https://www.segger.com/products/debug-probes/j-trace/technology/real-time-code-coverage/.

[36] G. Cover, "G Cover," 10 2017. [Online]. Available: https://www.ghs.com/products/safety_critical/gcover.html.

[37] A. H. Patil, N. Goveas and K. Rangarajan, "Test case Autogeneration code Git hub repository," 10 2017. [Online]. Available: https://github.com/Abhinandan1414/CoojaTestCaseGeneration.

[38] D. Adam, "Full tcp/ip for 8 bit architecture," in *Proceedings of the first ACM International Conference on Mobysis*, Sanfransisco, 2003.

[39] "Raspberry Pi wikipedia," 8 10 2017. [Online]. Available: http://en.wikipedia.org/wiki/Raspberry_Pi.

[40] P. a. P.E.Black, "Abstracting formal specifications to generate software tests via model checking," in *Proc. 18th Digital Avionics System Conference*, 2013.

[41] R. Jain, The art of computer systems performance analysis, Wile-InterScience, 2010.

[42] E. Kreyszig, Advanced Engineering Mathematics, 2011.

[43] B. S. Grewal, Higher Engineering Mathematics, 2014.

IJSER

IJSER © 2019
http://www.ijser.org

[44] "JVM Hotspot command line arguments," 2017. [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html.

[45] I. Muneer, *Systematic Review on Automated Testing Types, Effort and ROI,* PhD Thesis, 2014.

[46] B. Ovilio, *Test effort and test coverage: correlation analysis in a safety critical operating system,* PhD Thesis, 2012.

[47] "Test environment management best practices," 2017. [Online]. Available: http://www.softwaretestinghelp.com/test-bed-test-environment-management-best-practices/.

[48] K.-J. L. a. J.-J. Y. Tong-Yu Hsieh, "Test Efficiency Analysis and Improvement of SOC Test Platforms," in *16th IEEE Asian Test Symposium*.

[49] J. V. Oenen, *Improving regression test code coverage using meta heuristics,* Thesis, 2010.

[50] A. H. Patil, "CodeCover: A Code Coverage Tool For Java Projects," in *ERCICA*, 2013.

[51] A. H. Patil, "CodeCover:Enhancement of CodeCover," *ACM SIGSOFT SEN,* 2014.

[52] S. B. a. R. Weber, "Enhancing Software Testing by Judicious Use of Code Coverage Information," in *IEEE Conference Publications*, 2007.

[53] R. M. K. a. R. Skibbe, "On software reliability and code coverage," in *IEEE Conference Publications*, 1996.

[54] M. H. a. R. M. H. Zheng Li, "Search Algorithms for Regression Test Case Prioritization," *IEEE transactions on Software Engineering,* 2007.

[55] J. O. P. Ammann, Introduction to Software Testing, Cambridge University Press, 2008.

[56] M. Y. T. O. L. Baresi, 2001. [Online]. Available: http://www.cs.uoregon.edu/michal/pubs/oracles.html.

[57] B. Beizer, Software Testing Techniques, Van Nostrand Reinhold, 1990.

IJSER © 2019
http://www.ijser.org

[58] *Introducing Combinatorial Testing in Large Organizations,* ASTQB, 2014.

[59] "Evaluating the t-way Technique for Determining the Thoroughness of a Test Suite," in *NASA IV&V Workshop*, 2013.

[60] "Combinatorial coverage measurement," in *NASA IV&V Workshop*, 2012.

[61] Y. Lei, "IPOG - A General Strategy for t-Way Software Testing," in *IEEE Engineering of Computer Based Systems conference*, 2007.

[62] "Test Management tools," 2017. [Online]. Available: http://en.wikipedia.org/wiki/Test_management_tools.

[63] K. A. A. a. S. G. Sergiy A. Vilkomir, "MIST: Modeling input space for testing tool," in *Proceddings of the 13th IASTED International Conference Software Engineering and Applications (SEA 2009)*, 2009.

[64] P.-L. Poon, "CHOC'LATE: a framework for specification-based testing," *Communications of the ACM 53.4,* pp. 113-118, 2013.

[65] L. B. a. M. Young, "Test Oracles," 2001. [Online]. Available: http://www.cs.uoregon.edu/michal/pubs/oracle.html.

[66] A. H. Patil, N. Goveas and K. Rangarajan, "CTT test execution log files," 05 Sept 2018. [Online]. Available: https://drive.google.com/drive/u/1/folders/1t_fT6rd3OLTfLiipmRyQi99_436emGAk.

[67] E. w. site, "Eclipse web site," 05 September 2018. [Online]. Available: https://www.eclipse.org/downloads/.

[68] A. H. Patil, N. Goveas and K. Rangarajan, "Test Suite Design Methodology using Combinatorial Approach for Internet of Things Operating Systems," *Journal of Software Engineering and Application,* 2015.

[69] A. H. Patil, P. Satish, N. Goveas and K. Rangarajan, "Integrated Test Environment for Combinatorial Testing," in *IACC*, Bengaluru, 2015.

[70] J. Bach and P. Shroeder, "Pairwise Testing - A Best Practice That Isn't," in *Proceedings of 22nd Pacific Northwest Software Quality Conference*, 2004.

IJSER © 2019
http://www.ijser.org

[71] Aranha, Silva and E. H. da, *Estimating Test Execution Effort Based on Test Specifications,* PhD Thesis, 2009.

[72] L. Tahat, B. Korel, M. Harman and H. Ural, "Regression Test Suite Prioritization using System Models," *Wiley Online Library,* 2011.

[73] Balcer, T. J. Ostrand and M. J., "The category-partition method for specifying and generating functional tests," *Communications of the ACM,* 1988.

[74] Behar, Y. Adler and N., "Code Coverage Analysis In Practice for Large Systems," 2011.

[75] Borba and E. Aranha, "Estimating Manual Test Execution Effort and Capacity Based on Execution Points," *International Journal of Computer and Application,* 2009.

[76] Bryce, R. C, S. Sampath, J. B. Pedersen and S. Manchester, "Test suite prioritization by cost-based combinatorial interaction coverage," *International Journal of System Assurance Engineering and Management,* pp. 126-134, 2011.

[77] Bryce, R. C and C. J. Colbourn, "Prioritized interaction testing for pair-wise coverage with seeding and constraints," *Information and Software Technology,* pp. 960-970, 2006.

[78] Chen and T. Yueh, "DESSERT: a DividE-and-conquer methodology for identifying categorieS, choiceS, and choicE Relations for Test case generation," *Software Engineering, IEEE Transactions on 38.4,* pp. 794-809, 2012.

[79] Clarke, J. Lawrence and S., "How well do professional developers test with code coverage visualizations? An empirical study," in *IEEE Conference Publications*, 2007.

[80] D. Kuhn, J.Higdon, J. Lawrence, R. Kacker and Y. Lei, "Combinatorial Methods for Event Sequence Testing," in *First Intl Workshop on combinatorial Testing*, 2012.

[81] R. Krishnan, S. M. Krishna and P. S. Nandhan, "Combinatorial testing: learnings from our experience," *ACM SIGSOFT SEN,* pp. 1-8, 2007.

[82] Dunkels, J. Vasseur and A., Interconnecting Smart Objects with IP The Next Internet, Morgan Kaufmann Publishers, 2010.

IJSER © 2019
http://www.ijser.org

[83] G. Rothermel, M. J. Harrold, J. Ostrin and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *ICSM '98: Proceedings of the International Conference on Software Maintenance*, 1998.

[84] Grochtmann, Matthias, J. Wegener and K. Grimm, "Test case design using classification trees and the classification-tree editor CTE," *Proceedings of Quality Week,* 1995.

[85] H.Washizaki and K. Sakamoto, "Open Code Coverage Framework: A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages," 2010.

[86] Harrold, J. A. Jones and M. J., "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Trans.Softw. Eng,* p. IEEE, 2003.

[87] Hildebrandt, A. Zeller and R., "Simplifying and isolating failure-inducing input," *Software Engineering, IEEE Transactions on, vol. 28,* pp. 183-200, 2002.

[88] Ghandehari, L. Shikh, Gholamhossein, Y. Lei, T. Xie, R. Kuhn and R. Kacker, "Identifying failure-inducing combinations in a combinatorial test set," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference*, 2012.

[89] Grag, F. D. Frate and P., "On the correlation between code coverage and software reliability," in *IEEE Conference*, 1995.

[90] L. Tahat, B. Korel, M. Harman and H. Ural, "Regression Test Suite Prioritization using System Models," *Wiley Online Library,* 2011.

[91] Thiagarajan and A. Srivastava, "Effectively prioritizing tests in development environment," in *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2002.

[92] S. Mirarab, S. Akhlaglv and L. Tahvildari, "Size Constrained Regression Test Case Selection using Multicriteria Optmization," *IEEE transactions on Software Engineering,* p. 2012.

IJSER © 2019
http://www.ijser.org

[93] McClary, C. J. Colbourn and D. W., "Locating and detecting arrays for interaction faults," *Journal of combinatorial optimization,* 2008.

[94] M. Lyu, J. Horgan and S. London, "A coverage analysis tool for the effectiveness of software testing," *IEEE Trans. on Reliability,* 1994.

[95] Qi, W. E. Wong and Yu, "Effective Fault Localization using Code Coverage," in *IEEE Conference Publications*, 2007.

[96] Qu, Xiao, M. B. Cohen and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization," in *ICSM 2007. IEEE International Conference on. IEEE*, 2007.

[97] R.Mercer, T.W.Williams and M., "Code Coverage, what does it mean in terms of quality," in *IEEE Conference Publications*, 2001.

[98] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions,* 2014.

[99] Yilmaz, Cemal, M. B. Cohen and A. A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *Software Engineering, IEEE Transactions on 32,* 2006.

IJSER