Volume 6, Issue 11, November 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Attempt to Solve Automating the Functional Test Case Generation From Generic Test Design for Functional Wireless Simulator

[1]Abhinandan H. Patil, [2]Neena Goveas, [3]Krishnan Rangarajan
[1, 2] Department of CS & IS, BITS Pilani, Goa, India
[3] Department of CSE, CMRIT, Bengaluru, India

Abstract: ***This paper documents the attempt to solve the automation problem of functional test case generation from generic test design for functional wireless simulator. We took Automated Combinatorial Test for Software(ACTS) for generating the generic test design, Contiki Operating System Java simulator for wireless simulator.***

Keywords: CT; ACTS; CCM ; NIST; IoT; NuSMV

I. INTRODUCTION

Development of Operating Systems for IOT devices is a continuous process. In addition to issues to be addressed to by most software, there is an additional requirement to make the Internet of Things OS compatible to newer devices. This leads to a developer problem of developing regression tests suites on a continual basis. Since the chosen Operating System had several configurations combinatorial testing was explored as an option. The simulator tool of Contiki Operating System was case study for wireless simulator. The standard tools of National Institute for Standards and Technology (NIST) were used for generating the test design document.

II. PROBLEM TO BE SOLVED

First problem we are looking at is, we want to measure the effectiveness of CT for Internet of Things (IoT) Operating System (A case study). The quantification of effectiveness will be through well accepted traditional methodology of Code Coverage.

Secondly we want to improve the effectiveness of the regression test suite of Contiki Operating System. There will be two scenarios.

- Regression test suite is designed from scratch using the ACTS tool of NIST
- Additional test cases are added to the existing test suite by designing them using the CCM (Combinatorial Coverage Measurement) tool.

Contiki IoT Operating System is picked up as case study since it has various hardware configurations (Mote types) and input parameter values in the test cases of regression.

The regression test suite of Contiki is base lined for version 2.7 and 3.0. We started with Contiki version 2.7 to begin with which has fixed number of test cases in the regression test suite. We found that the regression test suite is missing the test design document which is an area for improvement. The test design can be modeled using ACTs (Automated Combinatorial Testing for Software) and CCM tools of NIST and the test cases can be generated. The generated test cases can be mapped to functional test cases (Runnable test cases in the environment under study) with some effort.

There will be three set of test suites:

- Base line test suite of Contiki which already exists.
- ACTs generated test suite which in turn is converted to functional test suite by following the process to be outlined later.
- CCM generated additional test cases which will be augmented with the existing test cases of base lined regression test suite to form CCM enhanced regression test suite.

In each case the code coverage in COOJA (The simulator tool of Contiki) is to be measured to quantify the effectiveness in terms of coverage. Although the coverage in Contiki instead of its simulator would have made more sense, the coverage data of simulator should still tell the effectiveness of CT.

III. RESEARCH HYPOTHESIS

The doctoral work will gather the data which quantifies whether the CT (ACTS and CCM tool combined) is effective for software such as Contiki Operating System used for IoT. We may uncover few functional bugs in Contiki or its simulator during the execution of test cases. We want to ascertain whether the ACTS tool indeed increases the code coverage for a given number of test cases in regression test suite. In second case where CCM is employed we know the

additional test cases will increase the coverage but we want to know a certain percentage of increase in combinatorial coverage translates into what percentage of increase in code coverage.

IV. CURRENT STATE OF THE WORK

To begin with, we base lined our idea on the basis of paper titled "Combinatorial Testing of ACTS: A case study" which is published in 2012 in IEEE fifth International Conference on Software Testing, Verification and Validation [1] in which ACTS tool (24637 lines of uncommented code) was evaluated using CT.

We have documented the work which is completed in the form of papers in few forums [2] through [6].

In [2] and [3] we studied Regression test suites in general. In [4] and [5] we gathered the data which is reference point. Paper [6] was theoretical attempt to have Integrated test environment for CT.

V. TASKS ACCOMPLISHED

Following tasks have been accomplished.

- High level test design in ACTS.
- Modeling of existing test suite which will act as input for the CCM tool to generate additional test cases. The special character "*" was found handy in modeling the existing test suite of Contiki.
- Generic engine (To be explained later) for auto generation of csc (functional test cases) files which is developed using the libraries exposed by COOJA code with very minimal tweaking. The coding was in core-Java.

VI. FUNCTIONALITY OF NEW TOOL

The new tool to be introduced will have following functionalities as depicted in fig 1.

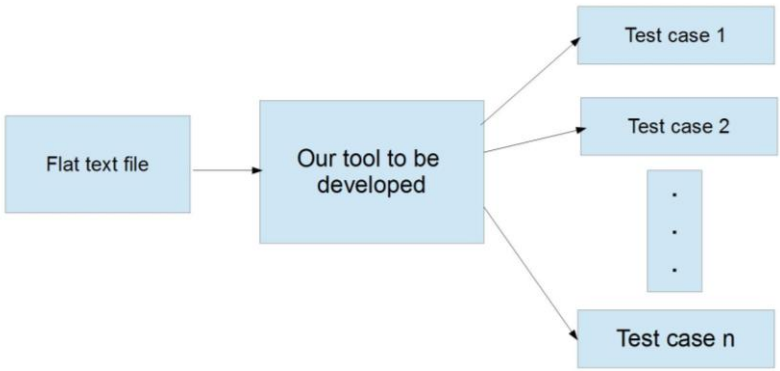

Fig 1. The functionality of our tool at very high level

```
[
{ testcasename,1 },
{ title , xyz },
{ speedlimit, xyz},
{ randomseed, xyz},
{ motedelay_us,xyz},
{ radiomedium,xyz},
{ events,xyz},
{ motetype,xyz},
{ motetype,xyz},
{ motetype,xyz},
{ mote ,xyz},
{ motetype_identifier,xyz},
{ mote,xyz},
{ motetype_identifier,xyz},
{ mote,xyz},
{ motetype_identifier,xyz}
],
[
],
.
.
[
]
```

Fig 2. Typical structure of human readable flat file

A flat text file which is more human readable can be parsed using the Regexp package of JAVA. The tool may have optional Graphical User Interface (GUI) for which template code is there. The GUI module is developed using the Swing and SWT.

The process flow is depicted in the fig 3 below. The human readable text file is parsed by parser developed using core Java. The functionality of this parser will be to extract logical blocks or records of text per test case. This information is used to populate the data structures of generic engine. The content of the data structure will act as a driver of the generic engine. When the generic engine is run it will output bunch of xml (csc) files. These csc files will be fully functional test cases except for the embedded java scripts. The embedded java scripts contain the scenario specific code. We have no intentions of automating the generation of java scripts as the effort is far more compared to returns. The final output of this whole process depicted in fig 3 is functional test cases which will run in Contiki test environment.

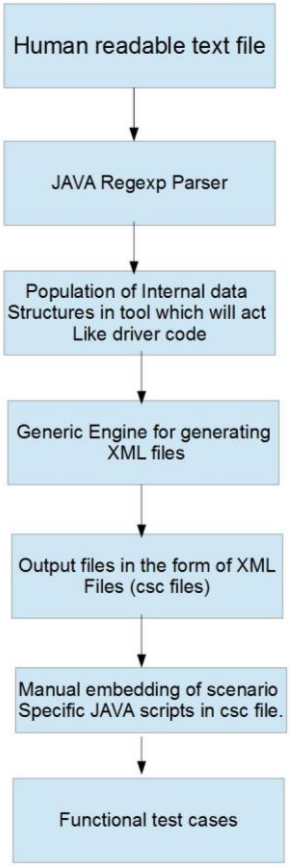

Fig 3. Process flow for generating functional test cases

The output of each stage becomes input for the next stage in fig 3. At the black box level human readable flat text file is the input and functional test cases are the final output of the whole process.

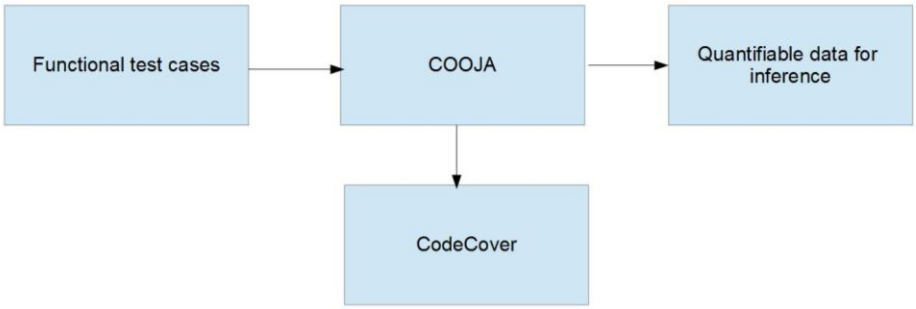

Fig 4. The process of gathering the code coverage data

The CodeCover tool is attached to the simulator tool of the operating system to gather the required information for inference.

VII. KNOWN LIMITATIONS AT THIS TIME

Following are the known limitations:

- We are facing the limitations of ACTS and CCM tool functionality for our kind of testing. ACTS output should just contain two columns in the output when required. Please have a look at the Appendix A of paper [4]. Mote type should mandatorily be present in the first column and second optional parameter for all the test cases.
- The coverage data of Operating System rather than its simulator would make more sense. We are aware of this.
- We want just two parameters in the ACTS generated design for our testing for two reasons. Existing java scripts embedded in the csc file are taking just two parameters at a time but still quite lengthy (LOC). If the ACTS

output is mapped as it is into functional test cases after dropping the irrelevant parameters for a given test cases, the length would be impractical. Only two parameters are manageable in our case.

- We looked at non NIST tools such as "ALLPAIRS". This is an interesting tool developed using perl. This tool does not meet our requirement.
- We modeled the existing test suite in CCM tool with 53 test cases and 9 parameter using the "*" character. We generated the missing test cases for 95% of required combinatorial coverage using the required additional test cases as 25 for 2 way interaction. The CCM output is gathered. Here again we encountered the same problem of unmanageable length of csc files.
- To get the required two column output in ACTs tool one could advocate to use the constraint feature. But the constraints list would be so long and unusable if one desires to maintain the test design.
- In the absence of NIST feature, we will be made to pick two parameters (Mandatory and optional) using heuristic which will be just an approximation of actual test design and our results may be misleading.

VIII. GENERAL OBSERVATIONS

Automation is for minimizing the repetitive work. The coding of Java scripts to be embedded in the individual test scripts has to be manual by understanding the specific scenario. Since we are studying the lowest most layer of IoT system (Operating system for IoT), the study is relevant to IOT systems. We tried to employ NuSMV (New Symbolic Model Verifier) along with ACTs for modeling the test design but found NuSMV unsuitable for this project. We tried Contiki and its simulator as IoT Operating System and functional simulator but found them unsuitable. NIST tools don't meet our requirements for the work mentioned in this paper.

IX. OUTCOMES OF DOCTORAL WORK

- Studying the automation of functional test case generation from the generic test design.
- Insight into functioning of IoT systems and IoT OSs.

X. EVALUATION OF RESULTS

Evaluation of our approach was not possible due to limitations of few tools employed. We have understanding of how to automate the functional test case generation from generic test design.

XI. SUPPLEMENTARY INFORMATION

Following additional information will be provided on demand.

- Test design
- Code
- Preliminary data gathered

ACKNOWLEDGMENT

We would like to thank open source software and free software for making this study possible.

REFERENCES

[1] Mehra N.Borazjany, Linbin Yu, Yu Lei,Raghu Kacker and Rick Kuhn, "Combinatorial Testing of ACTS: A Case Study,"IEEE Fifth International Conference on Software Testing, Verification and Validation,2012,pp 591-600,doi: 10.1109/ICST.2012.146

[2] 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

[3] 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

[4] 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

[5] 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

[6] 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

[7] D. Richard Kuhn, Raghu N. Kacker and Yu Lei 2013. Introduction to combinatorial testing, Text book.

[8] D. Richard Kuhn, Raghu N. Kacker and Yu Lei 2013. Practical combinatorial testing manual, NIST special publications, 800-142

[9] NIST, http://csrc.nist.gov/groups/SNS/acts/index.html

[10] C Nie et al, A survey of combinatorial testing, ACM Computing Surveys, Vol. 43, No. 2, Article 11, Publication date: January 2011.

[11] P. Ammann, J. Offutt, Introduction to Software Testing, Cambridge University Press, New York, 2008.
[12] B. Beizer, Software Testing Techniques, Van Nostrand Reinhold,New York, 2nd edition, 1990.
[13] J. Bach, P. Shroeder, Pairwise Testing - A Best Practice That Isn't. Proceedings of 22nd Pacific Northwest Software Quality Conference, 2004, pp. 180-196
[14] L. Baresi, M. Young, Test Oracles, Dept. of Computer and Information Science,Univ. of Oregon, 2001.http://www.cs.uoregon.edu/michal/pubs/oracles.html
[15] Combinatorial coverage measurement NASA IV&V Workshop Sept 11-13, 2012 Digital Avionics Systems Conference, Oct.1999, IEEE, vol. 2. pp.10.A.6.1-10
[16] Y. Lei, "IPOG - A General Strategy for t-Way Software Testing", IEEE Engineering of Computer Based Systems conference, 2007.-describes generalized IPO algorithm for constructing t-way covering arrays.
[17] Evaluating the t-way Technique for Determining the Thoroughness of a Test Suite, NASA IV&V Workshop, Sept.2013

