https://doi.org/10.48048/tis.2022.3472

Test Case Generation for Arduino Programming Instructions using Functional Block Diagrams

Mani Padmanabhan

Faculty of Computer Applications, Vellore Institute of Technology, Vellore, Tamil Nadu, India

(Corresponding author's e-mail: mani.p@vit.ac.in)

Received: 10 November 2020, Revised: 18 May 2021, Accepted: 28 May 2021

Abstract

Interconnecting different wireless sensor and actuator network in the real-time systems are more demanding for testing. High capability is needed to enable efficient methodologies for testing. The major source of effective testing is the identification of test cases. Programming instruction based test case generation has not well suitable for Arduino real-time systems, that an open statement in the research community. This paper introduces a functional block diagram based test case generation framework to facilitate the functional evaluation of Arduino real-time systems. It makes from the functional block diagram. First, block diagrams are converted to the Event Sequence Graph, then the event node is minimized, the test cases are generated minimized event. The presented methodology has validated with the Arduino programming language. This proposed approach quantified with five sensors based Arduino real-time system experiments shows that based on the validated test cases; the development time and cost of the Arduino real-time systems have dynamically reduced.

Keywords: Software testing, Test case generation, Functional block diagrams, Software validation, Software test path generation

Introduction

The COVID 19 pandemic condition shows the essentials of human-less automatic technology in all the sectors. Arduino based real-time system development is a major activity in 2020. Most of the electronics projects are building based on the open-source Arduino. The Arduino IDE (Integrated Development Environment) possible to access both a physical programmable circuit board or microcontroller and software. The Arduino physical board has compatible to upload the programming instruction from the computer [1]. Arduino software development based on system design. The major design for the Arduino based real-time systems is functional block diagrams. The automatic technology has to describe more in the development phase for the quality software programs. Producing high-quality software is becoming the main problem in real-time systems design due to the difficulty of real-time inputs [2].

The quality of the Arduino real-time system software measured based on the testing results. Testing is the process of identifying the bugs or error as much as possible or verify the process based on the input and output. The sensor-based Arduino real-time system is necessary to undertake effective testing to produce reliable systems. In many Arduino real-time system software development projects spending 50 % of the development time and effects in the error identification [12]. Most often, a tester is given a set of tasks based on the code for verifying the event that can be performed in the IDE. Software testing is the process of identify the bugs as much as possible meanwhile to reduce the cost of testing, test case selection in the regression testing is an essential activity to ensure the correctness of modified versions of software [23]. **Figure 1** shows the classification of Arduino software testing.

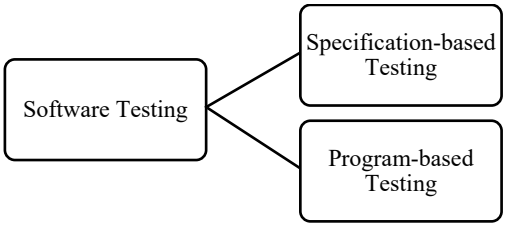

Figure 1 Classification of Arduino software testing.

A novel method to generate test cases based on a diagrammatic representation of the system behavior draws more attention in the research community. The detailed process of the system has produced functional block diagrams. The current testing methodology based on code-based. Code-Based Testing (CBT) generates test cases to test functionalities implemented in the Arduino IDE, so functionalities missing from software design will not be tested [25]. Thus, code-based testing tests what the source code does, rather than what it is supposed to do in real-time. A particularly important advantage of using functional block diagrams to drive test cases at a development level to support testing activities when source code is not available. A gap in the software test case generation research area exists that can be fulfilled with these approaches. **Figure 2** shows the major problem in the system-level testing process for sensor-based Arduino real-time systems.

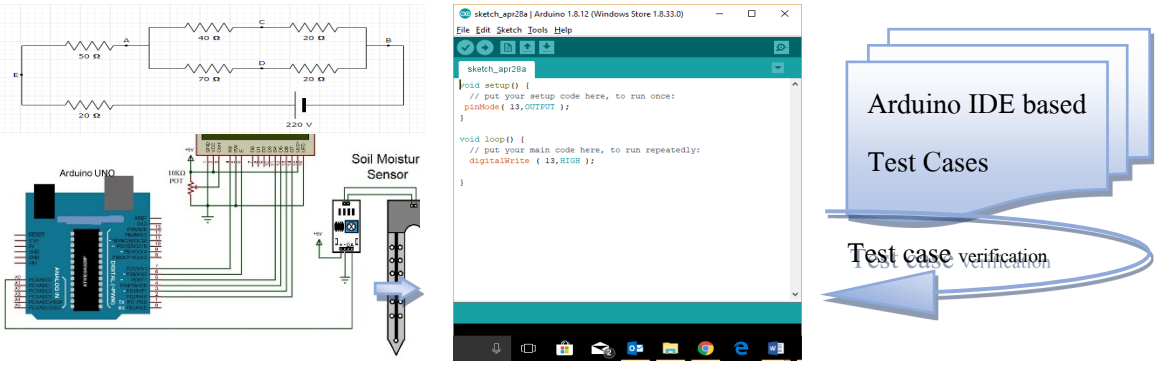

Figure 2 Test case verification using Arduino IDE.

The proposed methodology based on the functional block diagrams, the internal structures of the program requires during the software testing. In this approach has used the event graph and linking table for the test case generation then test cases are validated with internal structures of the Arduino programming. Associated test case generation researches are described in section 2. The proposed test case generation methodology with the Event Sequence Graph conversion algorithm is explained in section 3. The experiment on various real-time applications using the proposed approach in section 4. Conclusions are described in section 5. **Figure 3** describes the event linking table based test case generation. In the following section listed the related research based on the real-time systems.

Review of automated test case generation

This section delivers related evidence and concepts required to recognize the research work. In software testing, test cases are primary elements to test the programs. The purpose of test case generation is to reduce cost and human effort. Manual generation of large number of possible test cases and test data is a difficult problem whereas automatic testing can work round the clock and reduce tiring manual work. Mehrmand *et al*. (2011) emphasized that the most widely used area of testing is structural testing. It considers the unit under test as a white box, which means, test design is based on the internal structure of the program and requires programming skills. Mall *et al*. (2013) has presented the approach to generate

the code within class methods from UML. For this, they have built a novel graph model called sequence integration graph (SIG). In contrast to the conventional graph model (i.e., control flow graph), the SIG subsumes control flow graph and additionally contains method scope information of the interactions.

Wang *et al*. (2017) emphasized that continuous development of sensor based computer technology, safety-critical embedded systems were applied in many areas of more and more complex structure and function, once the failure of these systems will bring immeasurable social life and property losses. How to effectively improve the reliability of embedded software has become focus of by the industry and academia in general. Embedded software testing plays an important role in enhancing the credibility of embedded software systems, and model-based statistical testing has been widely used because of its high credibility and high efficiency at embedded process.

Table 1 Test case generations from diagrammatic representation.

Author	Input Model	Method	Intermediate Model	Coverage Criteria
Cavarra *et al*. [19]	Class Diagram State Diagram Object Diagram	Traversal Function minimization	Formal Behavioral Description	Test graph All traces paths
Ramler *et al*. [15]	State Diagram	BFS Traversal	State transition table	Class level testing
Samuel *et al*. [21]	Activity Diagram	Bottom-up Testing Strategy	Condition classification tree	Class methods
Wang *et al*. [18]	Sequence Diagram	DFS, Category partition	Scenario tree	Event, path, conditions

Ramler *et al*. (2012) presented the approach for test cases generation. In this research construct use case specification model and sequence diagrams of the SUT. The finite automata, and Event deterministic finite automata from sequence diagrams.

In proposed research has work differs from the above-mentioned research on the following grounds: A unified form of event flow instruction for block diagram, demonstration of block diagram based test case generation technique, comparison between block diagram based and code-based teat case generation approaches with regard to the accuracy, safety and efficiency. However, the approach tackles the challenges of test case validation of Arduino software.

Proposed approach

The objective measurement of test case quality is one of the key issues in software testing and has been a major research area for the last 3 decades. In this section, a brief overview of the Arduino block diagram notation and test case generation technique for sensor-based systems has been presented. The Arduino block diagram describes the control flow between input variables and output variables [1].

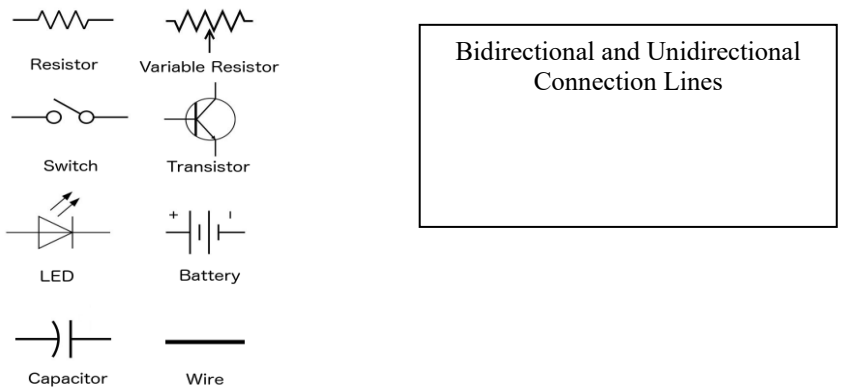

Figure 3 Basic symbols of Arduino block diagrams.

Arduino block diagram contains the common circuit diagrammatic representation even if the complex systems. Arduino block diagrams are associated with a resistor, switch, transistor, LED, battery, capacitor and wire. Arduino block diagrams are usually drawn to flow directional arrow. They define what functions occur, in which order those functions can occur, and what alternative paths of execution are available. **Figure 3** provides basic symbols and more common electronic components in Arduino block diagrams.

The sensor-based Arduino block diagrams to be the additional schematic symbols to show the unique properties of the event in the Arduino IDE. The sensor events are combined in the command line of the Arduino programming. The current test case generation methodology based on the functions and command line in Arduino programming. The methodology has well suitable for the system level test cases generation technique and validation process. In the system level validation may be more bugs and mismatching results if the sensor and real-time events are not collected during the programming development.

Programming instruction based test cases are verification of process in the real-time system but functional block diagram based test cases are a validation process. In the proposed technique to generate the test cases using Arduino block diagrams. First the approach covert the Arduino block diagram from the software requirements.

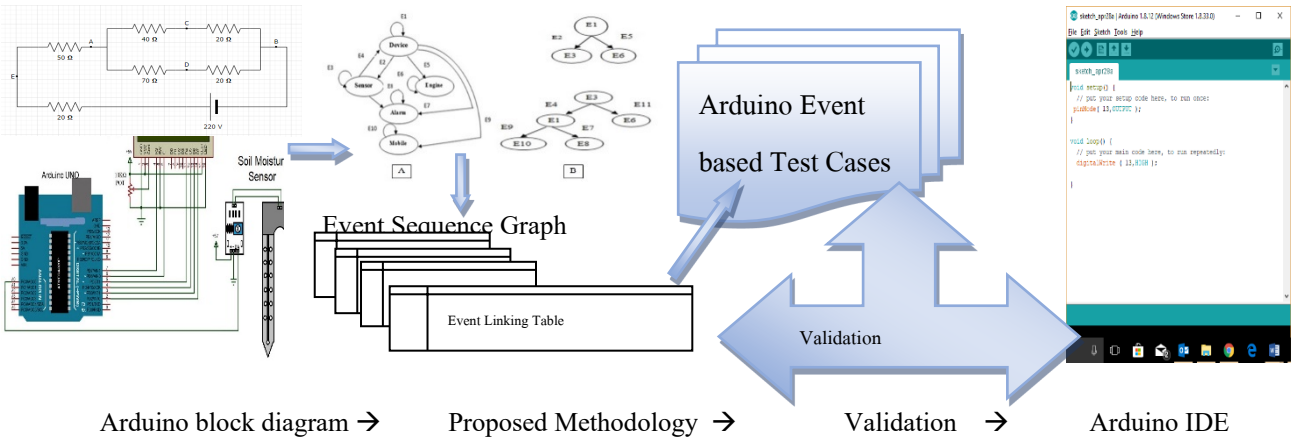

Figure 4 Proposed test case validation methodology.

In the next step, using the proposed algorithm identify the event and linking a binary graph. Based on the binary graph traversal operation select the block event items from the original graph to the new ordered graph, then 3 edge tree structures are minimized. In the final step, test cases are generated from the minimized event. The generated test cases are holding the input, process and output for the current event. The test cases are validated using post programming instruction. **Figure 3** describes the workflow of the proposed methodology.

Test case validation is the essential process of sensor-based Arduino real-time system development for assuring the quality and reliability of updated software. Test case validation based on the event flow is the technique of system-level testing to reduce the time and effort required for system-level testing. The proposed validation methodology double way process to enhance the efficiency and effectiveness of the Arduino RTS. In the subsection describe the Event Sequence Graph conversion from the block diagram.

Event Sequence Graph (EG) is a directed graph EG = ⟨V1, V2, E1, E2⟩ , where V is a set of nodes and E is a set of edges. The nodes in V are of 2 types: 'unidirectional' and 'bidirectional' where a 'bidirectional node' represents start or end. On the other hand, the edges are of 2 types: 'input edge' and 'output edge' the edges represent the change of method between 2 nodes. Algorithm 1 explains the conversion of sensor-based Arduino real-time system functional block diagram to the Event Sequence Graph. **Figure 4** shows the functional block diagram basic Arduino based tricolour LED. The wireless sensor-based signals are transferred to the tricolour LED based on the input red, green, or blue colour to be displayed in the common cathode RGB LED. The directional is described as the input. Algorithm 1 describes the Event Sequence Graph conversion steps from the Arduino real-time systems.

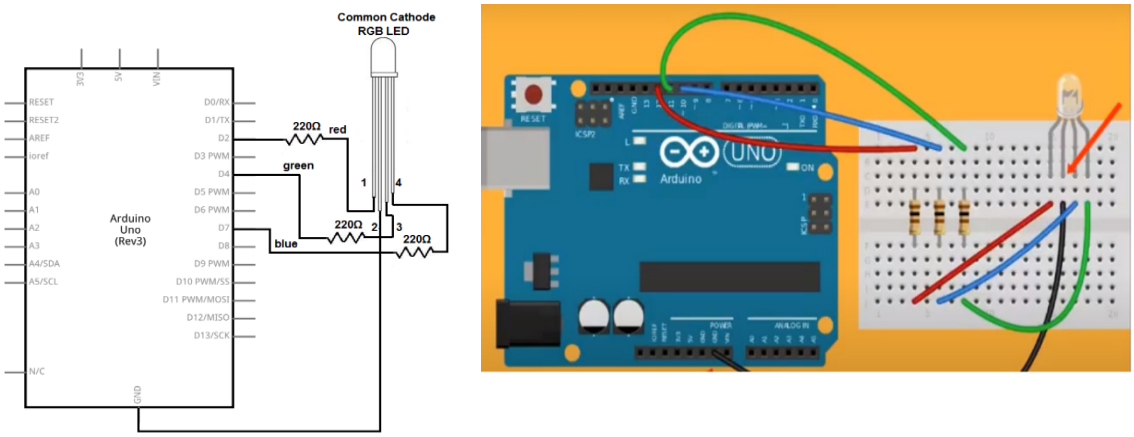

Figure 5 Arduino with tricolour LED functional block diagram.

The input value graph node has divided in D1 to Dn based on the direction, if the events are available in the block diagram then return the directed graph with an event or if the node is empty that is initial stage null then provide the empty graph with unique node. D1 allocation is enabled based on the direction either bidirectional or unidirectional. Finally, identity the sub node and proved the binary tree structure with left and right node.

Algorithm 1: Event Sequence Graph conversion

```
Pre          Graph node D1 contains direction of the event
Post         Directed graph EG or error returned
Return       Graph with node or null if no memory
if (memory not available)                    //initial stage (or) underflow
Graph == null
else
  Allocate (event)                  = 0
        Graph node D1 → count              = event ++
 Graph node D1→  connection lines = -1
         Graph node D1→ Bidirectional or Unidirectional
end if
    if (event not line)                    //During allocation of the new stack.
       recycle (event)
       Graph node D1 = process
    elseif (Graph node D1 depends on process)
       allocate (Graph node D1→stimulus process)
```

```
    else
       Graph node D1++
    reallocate (Graph node D1→stimulus)
    return Graph node D1 - Dn                    // Stack underflow

    Sub node [n] = Graph node D1(N);  //Identify the sub node
     If (D1(LN1) < D1(RN1)
          Graph node D1 [n-1] = EG(LN1)
       else
            Graph node D1 [n-1] = EG(RN1)
    End if;
End create Event Sequence Graph.
```

Figure 5 shows the generated Event Sequence Graph for tricolour LED based on the proposed algorithm. The top root node V1 is the sensor signal if no signal then off mode (right node) in the LED, In the V3, V4, V5 are the output signal based on the event. The directional from V2 to V3, V4, V5 are the bidirectional activity. Finally based on the graph event linking table formalized in **Table 2**.

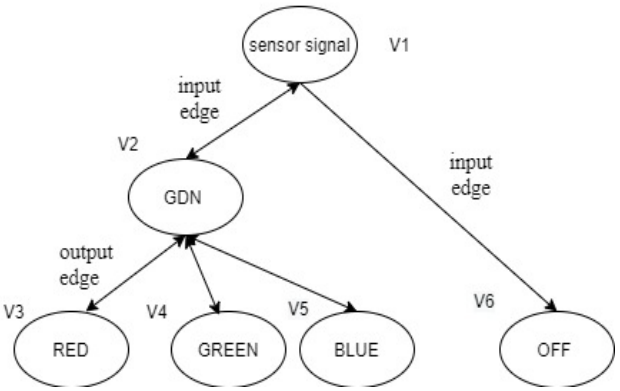

Figure 6 Event Sequence Graph for tricolour LED.

In the event linking table, the events are collected along with the information regarding the event direction either unidirectional or bidirectional, and input node or output node based on the algorithm-based event flow graph.

Table 2 event linking table.

Event ID	Event	Event direction	Node Value
T1	Signal to GND	bidirectional (E2)	input value (V1)
T2	Sensor Signal RED	bidirectional (E2)	output value (V2)
T3	Sensor Signal GREEN	bidirectional (E2)	output value (V2)
T4	Sensor Signal BLUE	bidirectional (E2)	output value (V2)
T5	Sensor Signal Off	unidirectional (E1)	output value (V2)
T6	Waiting sate	bidirectional (E2)	input value (V1)

The event attributes represent in data structure binary tree. In the binary tree, all the left nodes are input value (V1), the right nodes are output value (V2). If any bidirectional (E2) is minimized in the binary tree format as the unidirectional (E1), Finally V1, V2 of each event are ordered based on the pre-order travel the data collected and generated the event linking table.

The proposed event binary tree for tricolour LED, each tree pre-order value will produce the precondition, input value, output value, and post condition [7,11]. **Table 3** describes the generated test cases for the sensor-based tricolour LED.

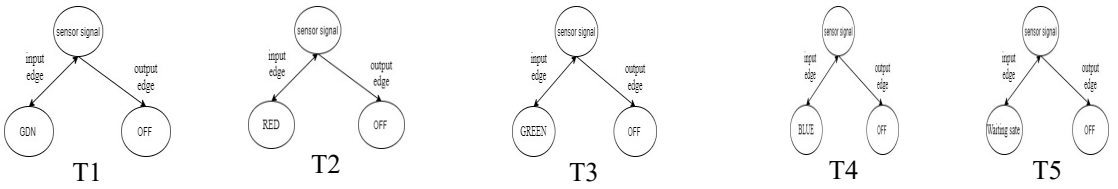

Table 3 Test case generation.

Binary tree ID	Event Scenario	Test Case ID	Precondition	Input	Output	Post condition
T1	Sensor Signal Off	Test Case1	Waiting sate (GDN)	Off command/ Equipment damage	LED off	Waiting sate (GDN)
T2	Sensor Signal RED	Test Case2	Waiting sate (GDN)	RED display	LED RED	Display mode
T3	Sensor Signal GREEN	Test Case3	Active mode/ Waiting sate (GDN)	GREEN display	LED GREEN	Display mode
T4	Sensor Signal BLUE	Test Case4	Active mode/ Waiting sate (GDN)	GREEN display	LED GREEN	Display mode
T5	Waiting sate	Test Case5	Display mode	Waiting sate (GDN)	Waiting sate (GDN)	Check Condition

The test cases are unique ID based on the binary tree, the left node and right nodes are the input and output for the root node. Precondition has provided the current state of the Arduino real-time systems. The post condition is the input from the Arduino real-time systems either through manual or automatic. Each binary tree produces the precondition, input data, and output value and post condition. Simultaneously, the methodology validates the event flow with Actual output, so that, the quality of test cases gets improved. The event linking table based test cases are identified the coverage criteria of the generated test cases. To illustrate the methodology, 4 different sensors based Arduino real-time systems are compared with the generated test cases in the experiments section.

Table 4 Test case validation.

Binary tree ID	Event Scenario	Test Case ID	Code based output	ELT based output	Actual output	Expected output
T1	Sensor Signal Off	Test Case1	LED off	LED off	LED off	LED off
T2	Sensor Signal RED	Test Case2	LED RED	LED RED	LED RED	LED RED
T3	Sensor Signal GREEN	Test Case3	LED GREEN	LED GREEN	LED GREEN	LED GREEN
T4	Sensor Signal BLUE	Test Case4	LED GREEN	LED GREEN	LED GREEN	LED GREEN
T5	Waiting sate	Test Case5	Not Applicable	Waiting sate (GDN)	Waiting sate (GDN)	Waiting sate (GDN)

Software testing for sensor-based Arduino real-time systems has been challenging due to the functionality of the current human-less requirements. The functional block diagram has been widely used as a system-level specification in Arduino real-time systems. The interaction diagram describes the system behaviour in the real-time process. However, due to a lack of efficient techniques for identifying simulated signals in Arduino real-time systems, the test case generation suffers from one of the major problems in software testing. An effective test case is necessary to produce reliable testing. Our proposed technique, describe the validation of test cases based on the event linking table and code-based. **Table 4** has described the process of validating the test case results with multiple constraints. The detailed experiment results with comparative results have recorded in section 4.

Results and discussion

This chapter summarizes the contribution of the present methodology in proposing a new technique for a more convenient and practical method of software test case validation through functional block diagrams of Arduino. The proposed approach validated the test cases generated by using an event linking table. The results obtained through experiments show that the proposed approach is feasible and capable of reducing the cost in SDLC. In path testing, the tester is expected to execute all paths of the test program. This testing style is very laborious and time-consuming. Program with loops have more number of paths and it becomes difficult to exercise all paths for testing purposes. To overcome this problem, a subset of events based on some criteria must be selected. The proposed block diagram approach provides all edge coverage criteria. **Figure 6** shows the source code coverage criteria based on several new test cases using functional block diagrams.

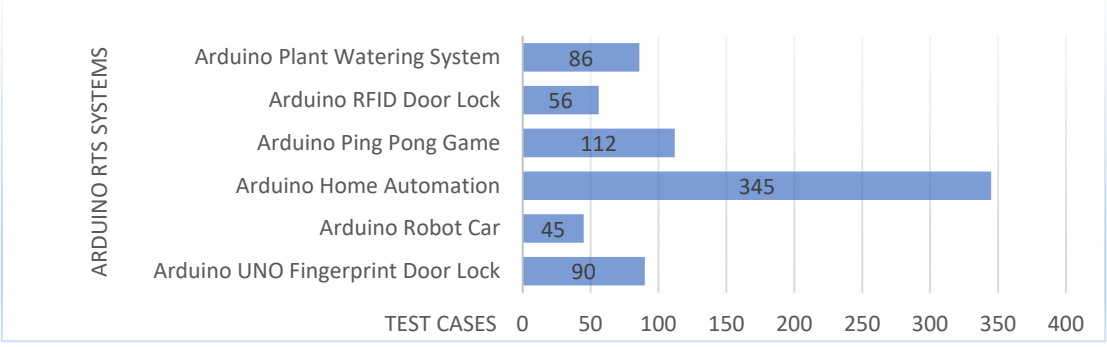

Figure 7 Generated test cases based on proposed approach.

To verify the performance of the proposed approach in this paper, 6 different prototypes Arduino real-time system experiments were conducted. During the testing, each event scenario produces the test cases. The coverage criteria to be calculated based on the path and branch coverage. The proposed test case generation for Arduino programming instructions using functional block diagrams yielded the effective results the comparative analysis of the proposed approach and the user acceptance testing has to be matched in **Figure 7**. In feature research investigating the possibility of automatic test case generation on the basis of other specifications in the functional block diagrams.

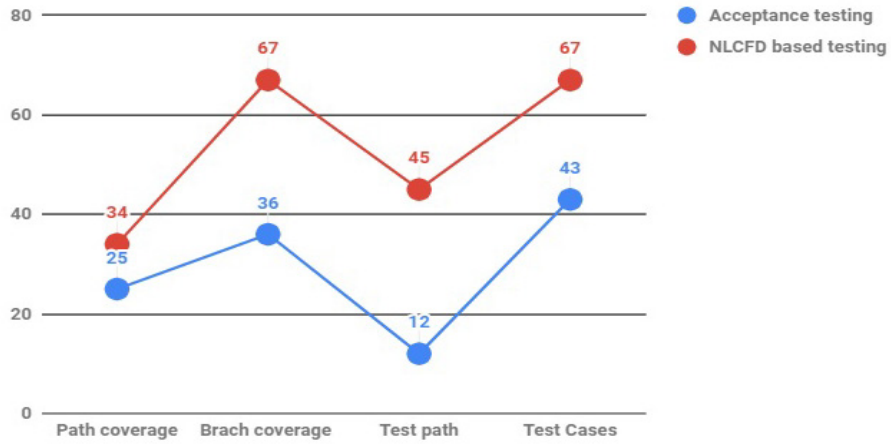

Figure 8 Comparative results of proposed approach.

It is observed that the proposed approach with validation technique, presented in these results, can generate test cases with high all-code coverage and can generate test data for given unique test cases. The validation approach shows the possible coverage in the block diagram based approach. Limitations of the technique observed during the experimental research and suggestions for future work are given in conclusions.

Conclusions

Software test case generation in regression testing is the process of checking the fault in the modified version. In the whole life time of the software, each version of composite functions must be tested to ensure the quality. Software test case generation is an important part of the software testing life cycle. It checks the functionality of the software concerning user requirements. This paper has deliberated Event Sequence Graph conversation algorithm, event linking table generation, and validation of test cases based on the functional block diagrams. Test cases are generated by exercising each part of the program with a suitable functional block in the Arduino real-time systems. The new faults have been detected with all path coverage in the proposed approach that has not been detected in the code based approach. In the future, the test case validation of the proposed work could be tried with other specification diagrams to fully automate the test case validation process.

References

[1] J Chen, L Zhu, TY Chen, D Towey, FC Kuo, R Huang and Y Guo. Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. *J. Syst. Softw.* 2018; **135**, 107-25.

[2] H Wang, J Xing, Q Yang, P Wang, X Zhang and D Han. Optimal control based regression test selection for service-oriented workflow applications. *J. Syst. Softw*. 2017; **124**, 274-88.

[3] R Mukherjee and KS Patnaik. A survey on different approaches for software test case prioritization. *J. King Saud Univ. Comput. Inf. Sci*. 2021; **33**, 1041-54.

[4] AP Agrawal and A Kaur. *A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection*. *In*: SC Satapathy, V Bhateja, KS Raju and B

Janakiramaiah (Eds.). Data engineering and intelligent computing. Springer Singapore, Singapore, 2018, p. 397-405.

[5] S Eghbali and L Tahvildari. Test case prioritization using lexicographical ordering. *IIEEE Trans. Softw. Eng*. 2016; **42**, 1178-95.

[6] R Feldt, S Poulding, D Clark and S Yoo. Test set diameter: Quantifying the diversity of sets of test cases. *In*: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation, Chicago, Illinois. 2016, p. 223-33.

[7] A Marchetto, MM Islam, W Asghar, A Susi and G Scanniello. A multi-objective technique to prioritize test cases. *IIEEE Trans. Softw. Eng*. 2016; **42**, 918-40.

[10] C Hettiarachchi, H Do and B Choi. Risk-based test case prioritization using a fuzzy expert system. *Inf. Softw. Technol*. 2016; **69**, 1-15.

[11] P Zhang, J Yu and S Ji. ADF-GA: Data flow criterion based test case generation for ethereal smart contracts. *In*: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Seoul, Republic of Korea. 2020, p. 754-61.

[12] M Padmanabhan. *Test path identification for virtual assistants based on a chatbot flow specifications. In*: KN Das, JC Bansal, K Deep, AK Nagar, P Pathipooranam and RC Naidu (Eds.). Soft computing for problem solving. Springer, Singapore, 2020, p. 913-25.

[13] M Azizi and H Do. Graphite: A greedy graph-based technique for regression test case prioritization. *In*: Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops, Memphis, Tennessee. 2018, p. 245-51.

[14] P Mani and M Prasanna. Test case generation for embedded system software using UML interaction diagram. *J. Eng. Sci. Technol*. 2017; **12**, 860-74.

[15] GH Subramanian, PC Pendharkar and DR Pai. An examination of determinants of software testing and project management effort. *J. Comput. Inf. Syst*. 2017; **57**, 123-9.

[16] P Mani and M Prasanna. Test case generation for real-time system software using specification diagram. *Int. J. Intell. Eng. Syst*. 2017; **10**, 166-75.

[17] P Mani. Test path identification for internet of things using transaction based specification. *In*: Proceedings of the International Conference on Current Trends towards Converging Technologies, Coimbatore, India. 2018, p. 1-6.

[18] H Wang, J Xing, Q Yang, W Song and X Zhang. Generating effective test cases based on satisfiability modulo theory solvers for service-oriented workflow applications: Effective test cases for service-oriented workflow applications. *Softw. Test. Verif. Rel*. 2016; **26**, 149-69.

[19] A Cavarra, C Crichton and J Davies. A method for the automatic generation of test suites from object models. *Inf. Softw. Technol*. 2004; **46**, 309-14.

[20] SK Ramler, DP Mohapatra and R Mall. Test case generation based on state and activity models. *J. Object Technol*. 2010; **9**, 1-27.

[21] P Samuel, R Mall and P Kanth. Automatic test case generation from UML communication diagrams. *Inf. Softw. Technol*. 2007; **49**, 158-71.

[22] C Wang, F Pastore, A Goknil, L Briand and Z Iqbal. Automatic generation of system test cases from use case specifications. *In*: Proceedings of the International Symposium on Software Testing and Analysis, Baltimore, Maryland. 2015, p. 385-96.

[23] P Mani and M Prasanna. Validation of automated test cases with specification path. *J. Stat. Manag. Syst.* 2017; **20**, 535-42.

[24] A Memon, Z Gao, B Nguyen, S Dhanda, E Nickell, R Siemborski and J Micco. Taming Google-scale continuous testing. *In*: Proceedings of the IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, Buenos Aires, Argentina. 2017, p. 233-42.

[25] H Srikanth, M Cashman and MB Cohen. Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study. *J. Syst. Softw*. 2016; **119**, 122-35.