An Embedded Software Testing Requirements Modeling Tool Describing Static and Dynamic Characteristics

Mingcheng Qu[1,2], Naigang Cui[2], Bingsong Zou[1], Xianghu Wu[1],
[1] *School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001*
[2]*Department of Aerospace Engineering, Harbin Institute of Technology, Harbin 150001*
Corresponding Author: Qu Ming-Cheng, E-mail: qumingcheng@126.com

Abstract

It is generally believed that Model-based testing (MBT) began in 1956. MBT is an important software quality assurance technology. We propose a set of graphical components describing the testing requirements of embedded software, and research how to plan test items and how to generate test cases. We couple some graphical components as a modeling system supporting to describe test requirement for embedded software. The modeling system proposed can enhance the efficiency of embedded software testing to a certain extent.

Keywords: Model-Based Testing; Automatic Test Case Generation; Test Requirement Modeling; Embedded Software

1 Introduction

It is generally believed that MBT began in 1956, initially mainly used for hardware testing, like hardware circuit and communication protocol and so on, and began to use for software testing by the end of the 70s. After years of research, MBT has formed three classical theories——automata theory, UML unified modeling theory and the combination of probability and statistics theory and automaton theory. These classical theories have their own representative models, including the finite state machine or the extended finite state machine, the UML models and the Markov chain.[1, 2, 3] Based on these classical theories and models, there are many MBT tools nowadays, like TVEC, Conformiq Test Generator, Reactis, Spec Explorer, UPPAAL-CoVer and AutoFocus etc.[4, 5, 6] However, these tools mainly support the classical models above, adopt classical methods to generate test cases, for example, the W method of FSM, cannot adequately describe the test requirements of the SUT(system under test).

In recent years, the research focus of MBT mainly includes putting the model into the software product line, test optimization, test evaluation, putting MBT into different fields (such as embedded systems, large-scale distributed systems and data acquisition system, etc.), test case generation from the UML models and other models (e.g., UPPAAL model and Simulink/Stateflow model, etc.).

The main contribution of this paper is that based on the classical theories and models, it proposes a model-based testing framework for general embedded software, puts forward a set of graphical system describing the testing requirements of embedded software, studies how to plan test items and how to generate test cases, and eventually develops a graphical test requirement analysis and software test case generation tool.

2 A Graphical System Describing Embedded Software Test Requirements

In this paper, the graphical system describing embedded software test requirements is composed of *data flow diagram (DFD), state transition diagram (STD), sequence diagram (SD), fault tree analysis (FTA), cause diagram (CD)* and *decision table (DT),* as shown in Fig. 1. Among them, the data flow diagram, the fault tree analysis, cause diagram and decision table are used to describe the static structure of software, and the state diagram and sequence diagram are used to describe the dynamic behavior of software. Besides, another big luminescent spot is the nested relationship between graphics. Data flow diagram can nest data flow diagram, state diagram, sequence diagram, the fault tree analysis, cause diagram and decision table. State diagram can nest state diagram, data flow diagram, the fault tree analysis, cause diagram and decision table. Fault tree analysis can nest fault tree analysis and decision table. All of the nested relationships are as shown in Table 1.

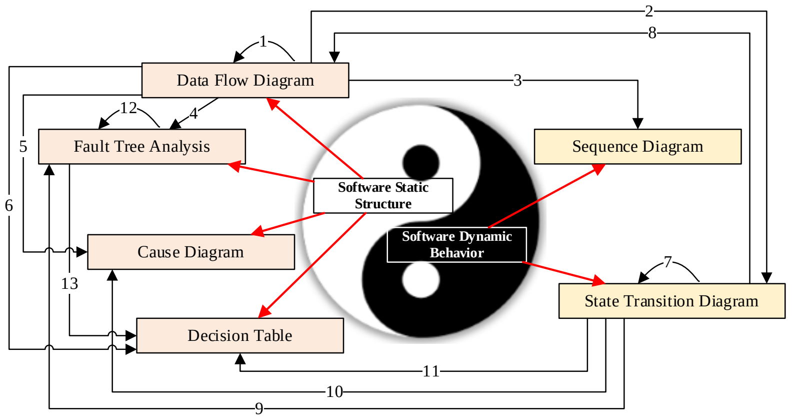


Fig. 1 A Graphical System Describing Embedded Software Test Requirements

Table 1 Graphics' Nested Relationship in the Graphical System

Parent Children	DFD	SD	STD	FTA	DT	CD
DFD	√	×	√	×	×	×
SD	√	×	×	×	×	×
STD	√	×	√	×	×	×
FTA	√	×	√	√	×	×
DT	√	×	√	√	×	×
CD	√	×	√	×	×	×

Data flow diagram, which is a logic description of embedded software's static structure, including the data flow and module partition, is a product of embedded software requirements analysis phase. DFD's graphical elements include data source, data target, data process, data storage, data flow and relationship of the data flow (and, or and xor), as shown in Table 2. Each graphical element has its own properties, which help to generate test cases. When creating the DFD, the data is managed by the global resources, including data variables, interrupts, hardware interface information and time domains, etc.

Table 2 Graphical Elements of DFD

Element Name	Data Source	Data Target	Data Process	Data Flow
Graphical Symbol	⬭	⬭	■	↗
Element Name	Data Storage	And Relation	Or Relation	Xor Relation
Graphical Symbol	⊏	*	+	⊕

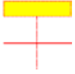
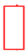
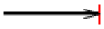
State transition diagram, which is a formal description of embedded software's dynamic behavior using states, is a tool for embedded software overall design and profile design. STD's graphical elements include initial state, final state, normal state, composite state, history state, state transition, condition node and entry/exit node, as shown in Table 3. Each graphical element has its own properties, which help to generate test cases.

Table 3 Graphical Elements of STD

Element Name	Initial State	Final State	Normal State	Composite State
Graphical Symbol	●	◉	■	■
Element Name	History State	State Transition	Condition Node	Entry/Exit Node
Graphical Symbol	Ⓗ	↗	Ⓒ	⊗

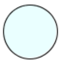
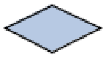
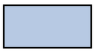
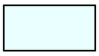
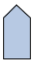
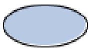
Sequence diagram, which is a timing description of embedded software's dynamic behavior using messages, is a tool for embedded software profile design. SD's graphical elements include object (lifeline), activation and simple message, as shown in Table 4. Each graphical element has its own properties, which help to generate test cases, especially the properties of the message.

Table 4 Graphical Elements of Sequence Diagram

Element Name	Object(Lifeline)	Activation	Simple Message
Graphical Symbol			

Fault tree analysis, which is similar to the error guessing method in traditional software black box testing, is a tool for embedded software testing design. FTA's graphical elements include and gate, or gate, basic event, transfer event, top event, gate event, switch event, condition event and connection, as shown in Table 5. Each graphical element has its own properties, which help to generate test cases.

Table 5 Graphical Elements of FTA

Element Name	And Gate	Or Gate	Basic Event	Transfer Event	Top Event
Graphical Symbol					
Element Name	Gate Event	Switch Event	Condition Event	Connection	
Graphical Symbol					

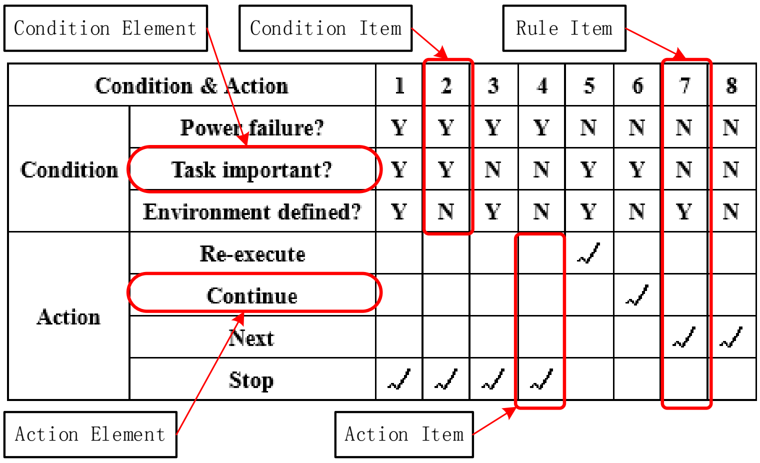
Cause diagram, which is similar to the cause and effect method in traditional software black box testing, is also a tool for embedded software testing design. CD's graphical elements include main cause, big cause, middle cause and small cause, as shown in Table 6. Each graphical element has its own properties, which help to generate test cases.

Table 6 Graphical Elements of Cause Diagram

Element Name	Main Cause(Red)	Big Cause(Blue)	Middle Cause(Green)

Graphical Symbol	🟥	🟦	🟩
Element Name	Small Cause(Gray)	Connection	
Graphical Symbol	⬜	↗	

Decision table, which is similar to the decision table method in traditional software black box testing, is another tool for embedded software testing design. DT's elements include precondition, condition element, action element, condition item, action item and rule item, as shown in Fig. 2. Decision table is easy to generate test cases, because the condition element can be seen as the input of test case and the action element can be seen as the output of test case.



Condition & Action		1	2	3	4	5	6	7	8
Condition	Power failure?	Y	Y	Y	Y	N	N	N	N
	Task important?	Y	Y	N	N	Y	Y	N	N
	Environment defined?	Y	N	Y	N	Y	N	Y	N
Action	Re-execute					✓			
	Continue						✓		
	Next							✓	✓
	Stop	✓	✓	✓	✓				

Fig. 2 Composition of Decision Table

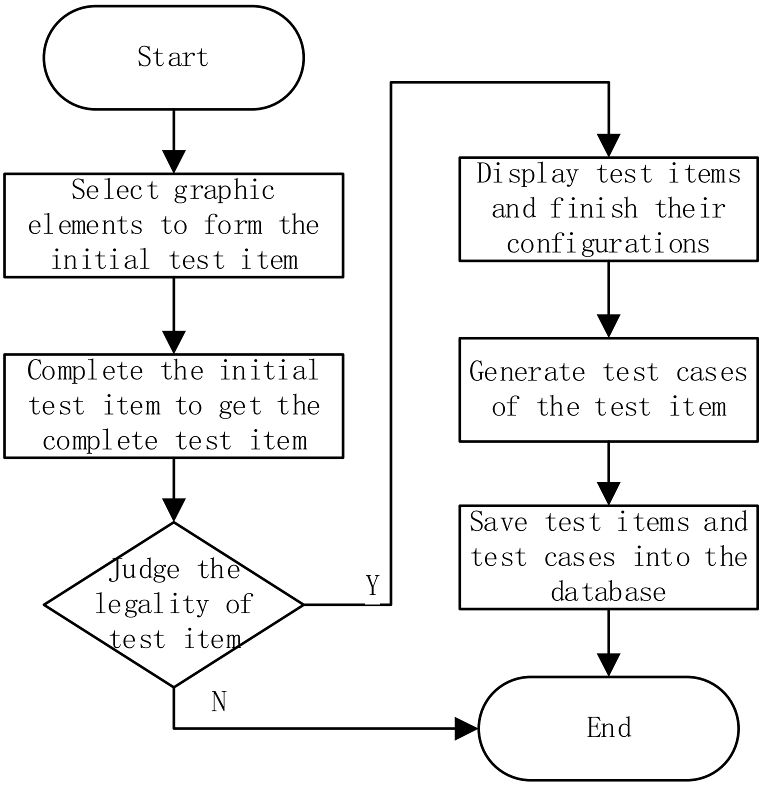

Start
Select graphic elements to form the initial test item
Complete the initial test item to get the complete test item
Judge the legality of test item
Y
N
Display test items and finish their configurations
Generate test cases of the test item
Save test items and test cases into the database
End

Fig. 3 Process of Test Item Planning and Test Case Generation

According to the description of the graphical system above, it can describe the static structure and dynamic behavior of embedded software, and combines a variety of classic methods in traditional software black box testing.

3 Test Item Planning and Test Case Generation

The process of test item planning and test case generation is as follows: (1) Select graphic elements to form the initial test items. (2) Complete the initial test item to get the complete test item according to certain strategy. (3) Judge the legality of test items (the test item is meaningful or not). If the test item is not legal, provide some prompt information, and the test item planning is over. (4) Display test items (current graphic elements' test item, sub-graph test item, decision table test item, time domain test item and so on). (5) Finish all the test items' configuration. (6) When test item's configuration completes, generate test cases of the test item. (7) Save the test items and test cases into the database. More clearly, the process is as shown in Fig. 3.

4 Summary

Model-based testing, which is a highly efficient software testing technology, can realize automatic testing and improve the efficiency of software testing to a great extent. In this paper we propose a model-based testing framework for general embedded software, puts forward a set of graphical system describing the testing requirements of embedded software and eventually develops a graphical test requirement analysis and software test case generation tool. The proposed modeling system can fully describe embedded software testing requirements, and supports automatic generation of more comprehensive test cases than traditional software testing methods.

References

[1] LI Shu-hao, WANG Ji, QI Zhi-chang. Model-Based Methods for Real-Time System Testing [J]. COMPUTER ENGINEERING & SCIENCE，2006(4)：119-123.

[2] Mlynarski, Michael, Gueldali, Baris, Engels, Gregor. Model-Based Testing: Achievements and Future Challenges[M]. San Diego: Elsevier Academic Press Inc., 2012: 1-39.

[3] YAN Jiong, WANG Ji and CHEN Huo-Wang. Survey of Model-Based Software Testing [J]. Computer Science, 2004(2): 184-187.

[4] Alan Hartman. Model Based Test Generation Tools[J]. Agedis Consortium, 2002.

[5] Muhammad Shafique, Yvan Labiche. A Systematic Review of Model Based Testing Tool Support[J]. Carleton University, Technical Report SCE-10-04, 2010.

[6] Ina Schieferdecker. Model-Based Testing[J]. IEEE Software, 2012(1): 14-18.