http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (5): 1038-1043, 2010
ISSN 1812-5638

Software Reliability Testing Data Generation Approach Based on a Mixture Model

[1]Wang Xin, [1]Han Feng-Yan and [2]Qin Zheng
[1]College of Electronics and Information, Xi'an Jiaotong University,
710049, Xi'an, Shaanxi, People's Republic of China
[2]College of Software, Tsinghua University, 100084, BeiJing, People's Republic of China

Abstract: To solve the problem about software reliability testing cases and testing data generation of real-time control systems, this study applies the reliability testing cases generation approach based on the mixture of operation profile and Markov chain which describes software operation profile by the use cases of UML, establishes the use model based on UML model for automatically deriving the testing model from the use model, generates a reliability testing case set based on the testing model and generates testing input data of reliability testing semi-automatically by eliciting input and output variables and abstracting testing input and output classes. The results of reliability testing on the actual airborne software show that the framework of the test case set generated by the proposed model is fairly stable. Thus, the proposed approach is suitable for generating reliability testing data of the real-time control system software semi-automatically, greatly simplifies the reliability testing process, improves testing efficiency and ensures testing validity.

Key words: Airborne software, software testing, reliability, data generation, operation profile, Markov chain

INTRODUCTION

Software testing is an important technique for software quality assurance (Mustafa *et al*., 2007). And software reliability testing is the key technology for improving and estimating software reliability which has the difficulty of generating test cases. There are mainly two types of models for generating test cases at present, namely the generation model based on operational profile and Markov chain. There are many researches after Musa (1993) proposed the concept of operational profiles including the expression of software component operational profile, test cases generation and the reliability evaluation based on operational profile testing (Woit, 1994), the approach for constructing the software operational profiles with synchronous and reactive characteristic (Ouabdesselam and Parissis, 1995) and the approach (Elbaum and Narla, 2001) for refinement methodology for the operational profiles suitable for different user mode and etc. However, these above approaches did not definitely consider the dependency relationships of various inputs and various times of the same input, thus it is difficult for them to be applied into software modeling of real-time control system with synchronous and reactive characters. The test case generation approach based on Markov chain includes the following two modeling approaches, namely the state layered Markov chain and the Markov chain based on the probability state diagram (Whittaker, 1993; Jame and Michael, 1994; Wohlin and Runeson, 1994; Shukla *et al*., 2004). However, with the growing of software complex, the state space of the Markov chain increases exponentially, thus makes Markov chain be difficult to model the complex software system. Therefore, although these two approaches lay a foundation for software modeling and reliability test case generation, there are still problems left which are difficult to be solved.

The reliability testing data are sent to the testing system through the interfaces of simulation testing environment. For the correctness of software reliability testing, the testing data must reflect the actual inputs of testing system realistically as well as the inputs of airborne embedded software should be described normatively for generating testing data.

This study describes an approach which synthetically uses the mixture model (Xin *et al*., 2007), constituted by operation profile and Markov chain model and utilizes the class chart of UML to generate software reliability testing data by eliciting input and output variables and abstracting testing input and output classes.

THE MIXTURE MODEL FOR TESTING CASES GENERATION

UML utilizes Use Case (UC) to describe the system functions, State Chart (SC) to simulate the dynamical operation condition of test cases, utilizes class chart to

Corresponding Author: Wang Xin, College of Electronics and Information, Xi'an Jiaotong University,
710049, Xi'an, Shaanxi, People's Republic of China

express the relationships of the class elements, utilizes sequential chart to express the sequences of mutual changing information between objects and etc. The modeling idea of the mixture model is using testing profile to define UC and using Markov chain to describe the dynamic characters of SC.

The construction of use cases and state chart

Constructing use cases

Definition 1: UC can be denoted by the following six-tuple array:

$$C_U = <I_{UcID}, C_{PreCond}, S_C, R_{OtherReq}, S_{PostCondSet}, p_{UC}> \quad (1)$$

where, I_{UcID} is the ID of use case, $C_{PreCond}$ is the prepositive condition of the case operation, S_C is the state chart which is corresponding to UC, $R_{OtherReq}$ is the other requirements with regard to this case, such as the requirement of the execution time of case; $S_{PostCondSet}$ is the set of postpositive conditions which express the state in which system enters after UC has been successfully executed; p_{UC} is the probability of the case operation.

The construction of UC can be realized by the following steps, such as confirming the starter of operation, defining the operation mode, constructing the operation list, defining the occurrence probability, calculating the occurrence probability and etc.

Constructing state chart

Definition 2: The SC of UC can be denoted by the following nine-tuple array:

$$S_C \leq S_{StateSet}, S_{tranSet}, \rho, S_{LabSet}, S_0, S_{FinishSet}, S_{EventSet}, S_{CondSet}, S_{actSet}> \quad (2)$$

where, $S_{StateSet}$ is the state set which is nonempty and finite; ρ: $S_{TranSet} \rightarrow 2^{StateSet}$ describes the dendriform hiberarchy relationship b etween the states; $S_{TranSet} \subseteq 2^{StateSet} \times S_{LabSet} \times 2^{StateSet}$ means the transition relationship; S_{LabSet} means the set of transition labels; S_0 means the set of initial states; $S_{FinishSet}$ means the set of terminal states; $S_{EventSet}$ means the set of trigger events; $S_{CondSet}$ means the set of monitor conditions; S_{ActSet} means the set of actions.

$\rho(s)$ defines the set of child states of state s. $\rho^*(s)$ means the set of child states including the state itself. $\rho^+(s)$ means the set of child states excluding the state itself, namely, $\rho^+(s) = \rho^*(s)-\{s\}$. For each state chart, there is a root state $r \in S_{StateSet}$. $\forall s \in S_{StateSet}$, $r \notin \rho^+(s)$, which means that there is no cycle in the hierarchy structure.

If X is given as the set of source states of transition, Y is given as the set of destination states of transition, L is given as the set of transition labels, then $\forall t = (X, L, Y) \in S_{TranSet}$, we have $X, Y \subseteq S_{TranSet}$, $X \neq \Phi$, $Y \neq \Phi$, $L \in S_{LabSet}$.

For transition $L \leq S_{LabSet}$, there is $L \leq E[c]/A>$, where, E is the trigger event, c is the monitor condition, A is the action. Only when c is true, E generates the action A, otherwise E does not generate actions. Transition L can be a simple transition or a subsequent transition. It may enter a combination state, or emerge from a combination state.

The output of operation model and testing model

Obtaining use chart from state chart: The key idea to convert the UML state chart into use chart is spreading state chart and eliminating the hierarchy of UML state chart. The hierarchy relationship of SC is mainly described by the OR state and AND state.

Definition 3: For one state s, its form can be a basic state, OR-state, or AND-state.

When s is a basic state:

$$\rho(s) = \phi$$

When s is an OR-state and $\rho(s) \neq \phi$, $\rho(s)$ is the OR decomposition of state s. When some object is in state s, actually it is in some child state of s.

When s is an AND-state and $\rho(s) \neq \phi$, $\rho(s)$ is the AND decomposition of state s. When some object is in state s, actually it is in all child states of s.

Definition 4: For one state s, if no special definition, the first child state s, in which entering first after entering the state s, will be called the default child state of s.

Definition 5: If $s_1 \in \rho^*(s)$, then we call s_1 the offspring of s and call s is the ancestor of s_1. If $s_1 \in \rho^*(s)$ and $s_1 \neq s$, then we call s_1 the strict offspring of s and call s is the strict ancestor of s_1.

The transition from SC to UC can be achieved according to the following principles:

- Spreading SC by recursively introducing the subordinate state chart which has been replaced by the replace label at present
- Normalizing the trigger
- Replacing the auto-transition by the transition with ε
- There should be no unreachable or endless state in use chart. If there is such a state in user chart, it should be corrected

Transferring UC to usage model: The state transition has the probability character in the usage model. For obtaining the usage model, the probability distribution of the software anticipative operation should be confirmed, namely the occurrence probability p(E) of the trigger event E and the true probability p(c) of the monitor condition c. The transition label $L \in S_{LabSet}$ can be extended to $L \leq sE(p(E))[c(p(c))]/A>$.

The transition probability can be calculated by the methods based on hypothesis, history data and estimation.

Expression of Markov chain of usage model

Definition 6: The usage model based on Markov chain can be denoted by the following five-tuple array:

$$C_M \leq S, \Gamma, \delta, q_0, F> \quad (3)$$

where, S is the set of the execution states, Γ is the set of the transition labels, $\forall t$, $t \in \Gamma$, then $t \leq Name,p>$, where, Name is the name of transition, p is the transition probability, δ is a function, δ: $S \times \Gamma \rightarrow S$, it can normalize the transition relationship between states of the operation process, q_0 is the initial state, namely the state in which software is before execution, F is the set of terminal states in which software enters when software stops executing. The Markov chain usage model of this case can be generated by normalizing the definition 5 of the UC usage model.

The algorithm for generating test cases

Definition 7: Test Case (TC) set can be denoted by the following five-tuple array:

$$C_T \leq S_{TcIdSet}, S_{PreCondSet}, S_{SequSet} S_{PostCondSet}, S_{RuleSet}> \quad (4)$$

where, $S_{TcIdSet}$ is the orderly label set of TC, $S_{PreCondSet}$ is the preset execution condition set of TC, $S_{SequSet}$ is the execution step set of TC, $S_{PostCondSet}$ is the set of anticipative execution results of TC, $S_{RuleSet}$ is the criterion set for deciding whether TC is successfully executed or not.

If we have the extended UC set of software system and the corresponding SC and Markov chain usage model set, the reliability test cases can be generated by the following algorithm.

Input: Extended UC set and its corresponding SC and Markov chain usage model set.

Step 1: Initialization which includes: (1) Setting the total generation number N of TC (2) The TC label counter Count = 1; (3) $C_T = \phi$

Step 2: Selecting a C_U randomly according to the occurrence probability

Step 3: Extracting the UC information and putting the UC information into TC according to the following sequence, namely (1) constructing a t according to the definition of TC and initializing t; (2) t.TcId = Count; (3) t.PreCond= C_U. PreCond; (4) t.PostCond = C_U.PostCond; (5) the decision criteria are extracted by t.Rule according to C_U.OtherReq and C_U.PostCondSet

Step 4: The TC execution sequence is generated by the following steps, namely (1) selecting the corresponding Markov chain C_M of C_U (2) randomly generating a TC by C_M (3) selecting the corresponding state chart s of C_U and obtaining the accurate information of each operation step from s (4) testing sequence can be combined by the accurate information of usage sequences and steps; (5) equating test sequence with $S_{SequSet}$

Step 5: Putting TC into test case set $C_T = C_T \cup \{t\}$

Step 6: Judging whether the test cases is smaller than N, if the test cases is smaller than N, then Count=Count+1 and turns into step 2, else terminating the test cases generation

RELIABILITY TESTING DATA GENERATION APPROACH FOR THE AIRBORNE EMBEDDED SOFTWARE

The airborne embedded software belongs to the real-time embedded software. Therefore, constructing the simulation testing environment and automatically generating testing data are needed in reliability testing of the airborne embedded software. The following steps describe how to generate testing data by the synthetical analysis.

Analyzing software documents: The input and output information for testing can be elicited from software documents, such as system task document, software requirement specification, software interface requirement specification and so on. For example, the information such as format of testing data, source, destination, type and so on can be obtained from software interface requirement specification.

Eliciting input and output variables: The transmission data in each interface are abstracted as the input or output logistic variables for obtaining the input variables of software testing. Then these input or output logistic variables are tailored and transmitted for obtaining the final input and output variables.

Because the system testing of the airborne embedded software is generally realized by the simulation testing environment and many inputs of the testing system can be generated by the simulation testing environment, we should not provide the inputs for all of the input variables. Therefore, we should regard the testing system and simulation testing environment as a whole for arranging the conformed input variables.

Abstracting input class: The input class can be defined as the set of the external input data types of the airborne embedded software system and be described by the state chart of UML.

There are two constructing principles of the input class: one is constructing the input class according to the external interface relationships of the airborne embedded software system; the other is constructing the input class according to the relationship of temporal logic.

- Constructing the input classes according to the external interface relationships

Constructing the input classes according to the external interface relationships can make the interface relationships of the testing system be corresponding to the input classes. The process of abstraction and construction is intuitive as well as the testing data can be generated only by organizing the data members of input classes into the requirement format. This proposed approach for constructing the input classes is suitable for some digital interfaces, such as the buses of MIL-1553B, ARINC429, ARINC629 and etc. It means the information of the data members of input classes can be directly obtained from the format of the data frame of the above buses.

- Constructing the input classes according to the relationship of temporal logic

Constructing the input classes according to the relationships of temporal logic uses the object-oriented method for the further encapsulation of the input variables from the viewpoint of the relationship between input time and logic and makes the input variables easy to be expressed and organized by the state chart of UML. The form of this constructing principle is similar to the first principle, is more suitable for some discrete input variables whose analog values and input values are not very correlative and makes the analog values which are coincident on time be encapsulated into the same input class.

Generating reliability testing data: Software reliability testing data can be divided into two forms, namely pre-generation and script-based generation.

The pre-generation testing data are deposited in the data-base or data document, automatically read by the testing system according to the testing requirement in testing and sent to the appointed interfaces. This testing data are static and we cannot dynamically change the input values of testing data according to the testing situation.

The script-based generation testing data are automatically generated by the testing script and can dynamically generate testing data and have higher flexibility.

The required testing data can be generated by synthetically utilizing the two above generation forms. It means that the relatively steady testing data can use the pre-generation form and the testing data which change heavily can use the script-based generation form. When using the testing scripts to generate testing data, the description of data class is used as the foundation of data generation.

CASE STUDY

For validating the actual results of the proposed model, we apply the proposed model into radar burst control airborne software reliability testing.

In software reliability testing, a simulation testing environment can be constructed by using the actual equipment as the operation platform of testing software and using the simulation system as the external interface equipment. For constructing the simulation testing environment, we use the scenario simulation software to generate the data of real-time scenario, develop the simulation software of radar detection, trace point process and inertial navigation system and construct the semi-physical simulation testing platform which composes a closed loop with the testing radar task computer system. The simulation testing environment for the software of radar burst control airborne is shown in Fig.1, the software of radar burst control airborne is contained in the radar system software (RSHC).

First, we can generate TC according to the approach proposed by the model. 19 UC of initial, burst generation, burst project and burst control and etc are generated for the burst control software, one instance of the 19 UC is described in Table 1. For calculating the occurrence probability of UC, we analyze the actual data collected by the radar work task record software to confirm the occurrence frequency of task and the occurrence probability of test case.

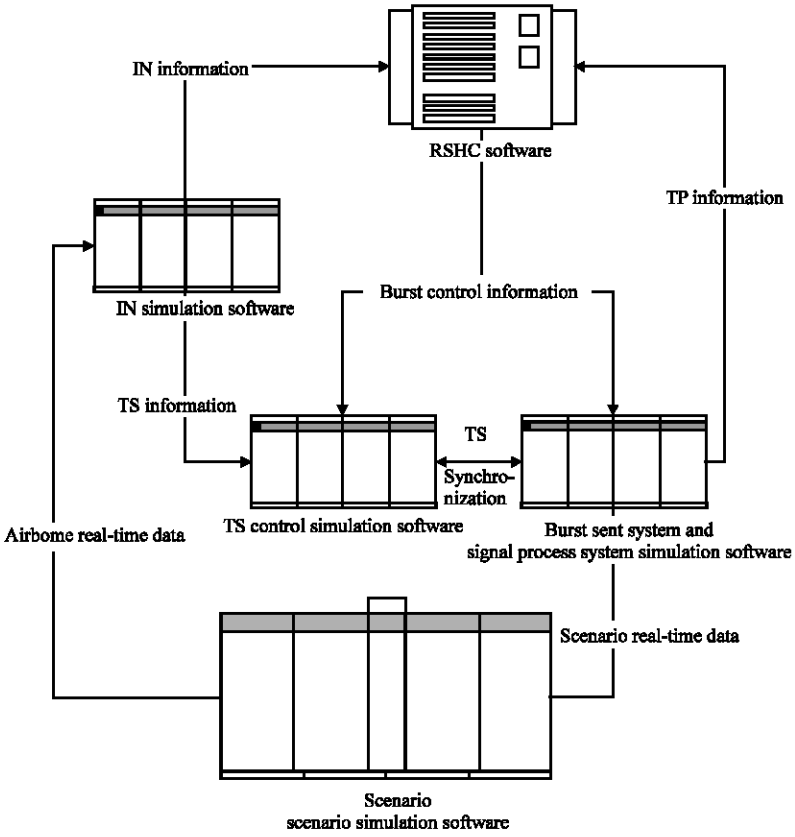

Fig. 1: The simulation testing environment for the software of radar burst control airborne

Table 1: The example of use case

Case ID	BD-002
Case name	The state management under work state
Objective	Transfer to the other states from work state
Actor	Operator
Pre-condition	The number of defined sectors is less than the maximum number of definable sectors when burst control under work state
Exit condition	Operator requests transferring the system state to stand-by state or emergent shutdown state
Invariant	None
Main success supposition	(1) Operator has open the operation dialog of radar state request (2) Operator has clicked the button of stand-by or emergent shutdown (3) Radar transfers to stand-by state or emergent shutdown state after receiving the state request
Variation	(1) Operator has clicked the button of emergent shutdown in the step 2 (2) Radar transfers to emergent shutdown state after receiving the state request.
Extension	(1) Radar doesn't respond the request when operator clicks the adjust button (2) Radar doesn't respond the request when operator clicks the maintenance button (3) Radar doesn't respond the request when operator clicks the shutdown button
Inclusive use case	None

SC is generated according to the listed contexts of the invariant, main success supposition, variation, extension, inclusive use case and etc. Then the Markov chain usage model of each UC can be obtained by spreading the SC.

Generating the testing data according to the above approaches and running three periods of system testing, generate more than one thousand testing data per periods and finally collect 37 failure data (Table 2). In the testing process, we use the supplementary tools developed by ourselves for model construction and testing data generation. The main testing process can be operated automatically.

The application results show that the TC set framework in each testing period generated by this model is basically stable and the specific TC are rarely repeated. Thus the reliability testing efficiency and validity of real-time control software can be improved by this model.

Table 2: The failure data of radar burst control software

Failures	Failure time (sec)	Failure interval (sec)	Failure	Failure time (sec)	Failure interval (sec)
1	100	100	20	36979	3151
2	2083	1983	21	38015	1036
3	3564	1481	22	43570	5555
4	3880	316	23	45101	1531
5	5337	1457	24	45800	699
6	6225	888	25	46902	1102
7	6999	774	26	49170	2268
8	8185	1186	27	51504	2334
9	9295	1010	28	55905	4401
10	15166	5871	29	59315	3410
11	18475	3309	30	65508	6193
12	19400	925	31	67890	2382
13	20135	735	32	70315	2425
14	21200	1065	33	74379	4064
15	23155	1955	34	76283	1904
16	23800	645	35	78320	2037
17	25210	1410	36	81924	3604
18	31548	6338	37	82990	1066
19	33828	2280			

CONCLUSIONS

The software reliability testing data generation approach proposed in this paper solves the problems that the traditional approaches based on operation profile are difficult to describe the synchronous and reactive characters of real-time control system and is suitable for reliability testing data generation of the embedded real-time control software.

The approach proposed in this paper can be used for generating the reliability testing data of real-time control system automatically. We formalize the reliability testing process by normalizing the software reliability testing data generation approach based on mixture model which was proposed in this paper for developing the corresponding supplementary tools and then achieve the automatic reliability testing of real-time control software.

The reliability testing data generation approach presented in this paper focuses on describing the elements of software dynamical action to solve the problems that the state space of Markov chain may increase by the exponential form. Thus, this approach decreases the number of testing data greatly as well as realizes the automatic reliability testing, which makes the testing efficiency improved.

ACKNOWLEDGMENTS

My Research Project was fully sponsored by the National Key Technologies R and D Program of China under Grant No. 2004CB719401.

REFERENCES

Elbaum, S. and L.S. Narla, 2001. A methodology for operational profile refinement. Proceedings of Reliability and Maintainability Symposium IEEE Computer Society, Jan. 22-25, Philadelphia, PA, pp: 142-149.

Jame, W. and T. Michael, 1994. A markov chain model for statistical software testing. IEEE Trans. Software Eng., 20: 812-824.

Musa, J.D., 1993. Operational profiles in software-reliability engineering. IEEE Trans. Software, 10: 14-32.

Mustafa, G., A.A. Shah, K.H. Asif and A. Ali, 2007. A strategy for testing of web based software. Inform. Technol. J., 6: 74-81.

Ouabdesselam, F. and I. Parissis, 1995. Constructing operational profiles for synchronous critical software. Proceedings of 6th International Symposium on Software Reliability Engineering, Oct. 24-27, Los Alamitos, USA., pp: 286-293.

Shukla, R., D. Carrington and P. Strooper, 2004. Systematic opertional profile development for software components. Proceedings of the 11th Asia-Pacific Software Engineering Conference, Nov. 30-Dec. 3, Los Alamitos, USA., pp: 528-537.

Whittaker, J.H., 1993. Markov analysis of software specifications. ACM Trans. Software Eng. Methodiol., 2: 93-106.

Wohlin, C. and P. Runeson, 1994. Certification of software components. IEEE Trans. Software Eng., 20: 494-499.

Woit, D., 1994. Operational Profile Specification, Test Case Generation and Reliability Estimation for Modules. McMaster University, Kingston, Canada.

Xin, W., Q. Zheng and H.F. Yan, 2007. UML based hybrid model for generation of software reliability test cases. J. Xi'an Jiaotong Univ., 41: 421-425.