in the GPS Signal Attenuation

Seongjin Cho, Sungkuk Cho and Byungkook Jeon

2425

Metamodeling for Automatic Test Case Generation

Bokyung Park, ByungKook Jeon, R. Young Chul Kim and Jae Hyub Lee

2431

Advanced Efficiency of DC-LED Lighting Systems to Replace the Conventional AC-LED Lighting Systems

Yeon Seok Lee, Sang-Ul Choi and Sherif A. S. Mohamed

2437

Development of Reader for Non-metallic Permeability Enhancement on 134KHz Bandwidth

Min Soo Kang, Yong Gyu Jung and Jae Ho Lee

2443

Relative Location Measurement in Container-Grown Seedlings Based on Wireless Sensor Networks

Sang-Hyun Lee and Kyung-II Moon

2449

Neural Networks based Adaptive Tracking for Nonlinear Systems

Raheel Quraishi, Nisar Ahmed and Muhammad Shafiq

2457

Analysis of Competitiveness in Secondary Battery Using Bibliometric Analysis

Young-il Kwon, Dae-hyun Jeong, Hyun-kyoo Choi and Jae Yun Lee

2477

Calculation of Synthetic Electric Field and Ion Flow Density along UHVDC Transmission Lines Based on the Non-Grid Method

Qinghui Zhang and Cui Yong

2489

Design and Evaluation of Self-organizing Scheduling Mechanism Adaptive To Traffic Load for Wireless Networks

Yoshiaki Taniguchi and Kyohei Inoue

2503

Virus detection backdoor in Microsoft Security Essentials

Silnov Dmitry Sergeyevich and Tarakanov Oleg Vladimirovich

2513

Cloud Storage Integration as a Learning Object Repository for Massive Open Online Course

Ridi Ferdiana

2521

Efficient Robust and Secure E-DRM with Encrypted Content Search

Maged Hamada Ibrahim

2531

Performance Comparison of Two Open-source CloudStack Client Libraries for High-performance Computing

Guohua Li, Hyeyoung Cho, Seung Chan Shin, Jaekeun Yeom, Jaegyoon Hahm and Sang Boem Lim

2547

Performance Analysis and Improvement of Meteorological Impact Assessment System

Tongji Jin, Ying-Hua Bian, Rongxu Xu and Sang Boem Lim

2561

The Evaluation of Vehicle Interior Acoustics Through Statistical Neural
Metamodeling for Automatic Test Case Generation

Bokyung Park*, ByungKook Jeon**, R. Young Chul Kim*, Jae Hyub Lee***

*SE Lab., Department of Computer and Information Communication, Hongik University, Sejong Campus, 339-701, Korea
E-mail: {park, bob}@xelab.hongik.ac.kr*
** Dept. of IT Eng., Gangneung-Wonju Nat'l University, Korea
jeonbk@gwun.ac.kr**
***Dept. of Computer Science and Engineering, Koreatech, Korea
jae@koreatech.ac.kr***

Abstract

The previous research suggested test case generation from Use Case [1], which made use case diagram and specification, identified use case scenario based on event flows, generated scenario matrices, and then pretend to extract test case. But it did not consider automatic mechanism for test case generation. We adopt metamodel approach to generate automatic test case. This approach easily transforms each model, and possibly develop automatic transformation. In this paper, we suggest to design metamodels for each model (that is, Use Case, Cause-Effect, Decision Table, and Test Case) needed to extract test case based on use case diagram.

Key Words: Use Case Approach, Meta-model, Cause-Effect Diagram, Decision Table, Test Case

1. Introduction

Currently as it gradually increases the complexity and scale of software, it is necessary for testing software to find error or defects [2]. Testing is sure to find error, and check whether requirements are reflected or not. It needs to make an effort at each development stage due on increasing cost to fix software developed with wrongly analyzing requirements. The rational software [1] also suggested to generate test cases from use cases, but not considered of an automatic tool. Kim & Son [3] suggested automatic test case generation based on state diagram with metamodeling and model transformation. In these tools, it is difficult to change if it does fix input model. To solve a problem, this paper mentions to adapt use case approach to extract test case. Use Case works at early stage of development lifecycle, and is defined requirements of software.

This paper is organized as follows: Chapter 2 explains related studies. Chapter 3 mentions to design metamodel for automatic test case generation. Chapter 4 the conclusion and future work.
2. Related Works

The rational software [1] used use case approach to generate test case, which represent use case diagram based on UML. Use case Specification is included Use Case Name, Flow of events (Basic Flow, Alternate Flow), Preconditions, and Postconditions. Flow of events is important to extract test case from use case. That is, it makes use case scenario based on flow of event, then extracts test case based on this scenarios. Kim [2] also suggested 'Metamodel oriented Automatic Test Case Generation Based on Transforming UML 2.4.1 Message-Sequence Diagram via Cause-Effect Diagram' as follows:

Step 1: For each use case, generate a full set of use case scenario and scenario matrix.
Step 2: For each scenario, identify at least one test case and the conditions that will make it execute.
Step 3: For each test case, identify the data values with which to test.

3. Metamodel for Extracting Automatic Test Cases based on Use Case

In this paper, we adapt metamodel mechanism to generate test cases from use case diagram, which is similar with the previous use case approach [1] and also applied other diagrams in UML[2,3]. We mention how to design metamodel for generating test cases.

3.1 Test Case Generation

![Diagram](image)

Fig. 1. Test Case Generation based on Use Cases adopted with Metamodel approach [4].

Figure 1 shows the procedure of test case generation based on use case approach. Our suggested test case generation consists of six steps as follows:

Step 1 (Use Case Diagram): Identify actor and use cases and draw use case diagram based on relationship with use cases such as extends and includes relationships.
Step 2 (Use Case Description): Describe the specification of use cases, and identify use case name, flow of events, preconditions, and postconditions as attributes of each Use Case based on the use case format [1].

Step 3 (Use Case Scenario): To generate test cases from use case, it is important to consider ‘flow of events’ which consist of the basin main flow and alternative flow. A use case scenario is a kind of an instance of use case or a complete path [1]. Therefore it identifies ‘flow of events’ of each scenario, and then make the combination of them.

Step 4 (Cause-Effect Diagram): Analyze the relationship between cause/effect/condition and the combination of events. That is, use Cause-Effect Diagram which is satisfied with 100% functional requirements with minimal test cases [2,3]. This diagram represents logical relationship through True/False of condition and Boolean expression between cause and effect.

Step 5 (Decision Table): This table represents the logical relationship expression between cause and effect. Each column of decision table means a test case. A test case can be combined with input values (such as true, false, and constraints) of input.

Step 6 (Test Case): Be generated with decision table via Cause-Effect Diagram. Test case consists of Input, Precondition, Test Condition, and Expected Result. Test Condition represents logical expressions with the combination of input’s conditions.

3.2 Metamodel Design

Metamodel of use case diagram is considered of attributes (such as Actor, Use Case, Include and Extends relationship) in Use Case. An actor has associated with one more use case, which has a relationship between Actor and Use Case, and also has Generalization, Inclusion and Extension relationship between Use Cases. We design metamodel of use case diagram included attributes of any relationships.
Fig. 2. Metamodel of Use Case Diagram [4]  

Fig. 3. Metamodel of Use Case Description [4]

Figure 2 shows the metamodel of Use Case diagram. In Use Case Description, Use Case Model consists of Use case and Condition. Use Case consists of Actor and FlowOfEvent which separates BasicFlow and AlternateFlow. Precondition is linked at the start point. Postcondition is linked at the end point. Metamodel of Use Case is difficult to use and complex, but exists the similar research [5]. Figure 3 shows metamodel of use case description. Cause-Effect Model is included all elements at the most top node. This model has Cause, Effect, and Connector. Cause is linked with the start attribute and effect also linked with the end attribute at a Connector as a line linked with Cause and Effect.

Fig. 4. Metamodel of Cause-Effect Diagram [2]

Figure 4 shows Metamodel of Cause-Effect Diagram. There are not defined the standard metamodel of Cause-Effect Diagram [3]. Therefore, we design metamodel with the name of this diagram.

The decision table represents as a table with the values of Cause-Effects Diagram. The condition of Effect is the result with the logical expression of Cause-Effects Diagram. The Cause consists of several inputs, and the Effect also consists of several outputs. The Value of
test case shows T(True)/F(False).

Fig. 5. Metamodel of Decision Table[2,3]  Fig. 6. Metamodel of Test Case[2,3]

Figure 5 shows Metamodel of Decision Table [2]. There are not defined the standard metamodel of Decision table [3]. Therefore, we design metamodel of the name of this table. The metamodel of Test Case ‘TestCaseModel’ may have several test cases. PreCondition means the condition of input, and TestCondition represents the logical expression of the combination of input conditions. ExpectationResult represents the value of expected results. Figure 6 shows Metamodel of Test Case. A Test Case includes TestCaseModel included with Precondition, TestCondition, and ExpectationResult.

4. Conclusion
In the previous automatic mechanism, it was used an algorithm approach. In this paper, we adapt Cause-Effect Diagram to generate test case based on use case diagram and scenarios, and also design each metamodel within each step. With Cause-Effect Mechanism, it is satisfied with 100% functional requirements with minimal test cases. It is necessary to use metamodel and model transformation for automatic test case generation, but not mention them in this paper.

5. Acknowledgments
This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2012M3C4A7033348) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2013R1A1A2011601), and also funded for Education & Research
Promotion of Koreatech.

References


***Corresponding author: Jae Hyub Lee, Ph.D., Professor
Department of Computer Science and Engineering,
Koreatech, Korea
E-mail: jae@koreatech.ac.kr***