# Combinatorial Test Architecture Design Using Viewpoint diagram

Yasuharu Nishi
The University of Electro-Communications, Tokyo
Tokyo, Japan
Yasuharu.Nishi@uec.ac.jp

Tetsuro Katayama
Faculty of Engineering, University of Miyazaki
Miyazaki, Japan
kat@cs.miyazaki-u.ac.jp

Satomi Yoshizawa
NEC Corporation
Tokyo, Japan
satomi@cj.jp.nec.com

***Abstract*— Software test has recently been a large-scale and complicated artifact, as is the software itself. It is necessary to reduce huge combinatorial test cases. This paper focuses on reduction of test parameters and combinations in test architectural design. First we will mention the test architecture design phase in TDLC: Test Development Life Cycle. Second we will introduce NGT: Notation for Generic Testing, which is a set of concepts or notation for design of software test architecture. This paper shows four examples of test architecture design patterns: Interaction-Viewpoint Conversion pattern, Interaction Cluster Partitioning Pattern, Interaction Demotion Pattern and Interaction Necessity Analysis.**

***Keywords- test architecture; test development life cycle; test viewpoint; combinatorial test; test design pattern; NGT;***

## I. Introduction

Software test has recently been a large-scale and complicated artifact, as is the software itself. There can be a test project with over one million test cases and over ten test levels including hundreds of test combinations. Technologies of large-scale and complicated software testing have just begun to advance and must be boosted.

For large-scale and complicated software testing it is necessary to reduce huge combinatorial test cases. It consists of three strategies:

*1) Test cases reduction*

Exhaustive test cases can be reduced by various combinatorial test techniques such as orthogonal array techniques and pairwise testing techniques. In this strategy test design focuses mainly on mathematical modeling of algorithms and constraints for fixed numbers of values in fixed numbers of parameters and combinations.

*2) Test values reduction*

Combinatorial test cases can be reduced by decreasing values such as equivalence partitioning. In this strategy test design focuses mainly on modeling of each parameter space for unfixed numbers of values in fixed numbers of parameters and combinations.

*3) Test parameters and combinations reduction*

Combinatorial test cases can be reduced by decreasing parameters and combinations. In this strategy test design focuses mainly on modeling of parameters and combinations directly for unfixed numbers of values in unfixed numbers of parameters and combinations.

These three strategies are all essential and can be applied simultaneously. Research based on the third strategy, however, is not active at present. This paper discusses modeling of parameters and combinations directly for combinatorial test design as test architecture design.

## II. Test Development Life Cycle

### *A. Test System Architecture and Test Suite Architecture*

"Software architecture" technology arose in the 1990s for development of large-scale and complicated software based on abstraction, separation of concerns, modeling, patterns and so on. "Software test architecture" technology is arising in our age, and we have to boost research and practices on software test architecture technologies more and more.

Architecture of software system has two kinds of scope: *system* architecture and *software* architecture. System architecture is for software, platform, peripherals, network et al. Software architecture is only for the inside of software, which mainly consists of modules (groups of statements) such as classes. Test architecture also has two kinds of scopes: test *system* architecture and test *suite* architecture. Test system architecture is for test system/software to be tested (SUT), platform where SUT is executed, generator of test cases et al. Test suite architecture is for the inside of test suites, which mainly consist of groups of test cases such as parameters of combinatorial tests, test conditions, test levels and test types.

There are several research and practices on test system architecture. UTP: UML Test Profile[1] is standardized as a notation based on UML for test system architecture. But currently research and practices of test suite architecture are just experiences and heuristics. In this paper hereinafter the word "test architecture" means test suite architecture. Fig.1 shows an example of test system architecture according to UTP. Fig.2 shows an example of test suite architecture according to NGT, Notation of Generic Testing[2] introduced in chapter III.

NGT can complement UTP because research and application of UTP mainly focus on test system architecture such as automation at present and NGT focuses on test suite architecture. NGT should harmonize UTP in future research.

### *B. Test Planning and Test Architecture Design*

Test process is recognized roughly by tradition as below: Test planning, test design and test execution. Traditional test design means a phase to derive test cases by test techniques such as control path testing. Traditional test planning means a phase which includes planning a test project and drawing a big picture of test cases, that is, which includes both tasks of the management side and the engineering side.

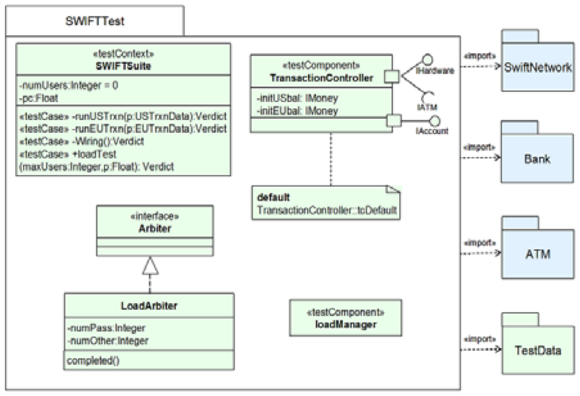


Figure 1. A Test system architecture example on UTP[1]

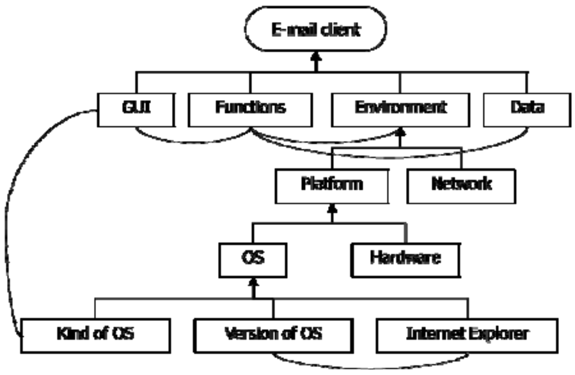


Figure 2. A Test suite architecture example on viewpoint diagram of NGT

A project planning phase in software development includes only tasks of the management side, while a software architecture design phase fills a role of drawing a big picture of software, that is, just the engineering side. A lot of companies separate positions of project manager and software architect. In software testing, however, tasks of both sides are traditionally mixed as test planning, test strategy or test approach, because software testing is a tight task for budget and effort constraint. A lot of companies have only a position of "test manager" for both sides while very few companies have a position of "test architect".

To boost research and practices on software test architecture technologies, we have to distinguish the management side and the engineering side. It is necessary to re-define test processes only from the engineering side named TDLC, Test Development Life Cycle. Fig. 3 shows TDLC, which consists of four phases: test requirement analysis, test architecture design, test detail design and test implementation. TDLC is intended just to develop test cases or test scripts. Whole test processes need a test execution phase, a test result recording phase and several test management tasks.

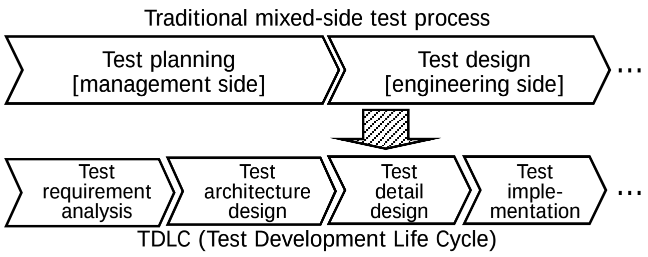

Traditional mixed-side test process
Test planning [management side]
Test design [engineering side]
…
Test requirement analysis
Test architecture design
Test detail design
Test imple-mentation
…
TDLC (Test Development Life Cycle)

Figure 3. TDLC (Test Development Life Cycle)

### C. *Test Architecture Design for combinatorial test design*

Combinatorial test design generally consists of three phases: 1) selecting parameters and combinations, 2) modeling parameter space and 3) determining and applying a combinatorial test technique such as orthogonal array techniques and pairwise testing techniques. The selecting parameters and combinations phase is operated with heuristics, experiences and engineering sense. We categorize the selecting parameters and combinations phase into test architecture design and the other two phases into test detail design.

## III. TEST ARCHITECTURE DESIGN

### A. *Concepts for Test Architecture*

As there is still no agreement on the exact definition of the term "software architecture", it is impossible to exactly define the term “test architecture” for the present. For example IEEE std. 1471[3] defines "architecture" as "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution". To follow IEEE's definition, we have to clarify components and their relationships for software test architecture as well as program statements for software testing.

It is natural for program statements to correspond to test cases or test scripts. This correspondence leads components to be group of test cases such as parameters of combinatorial test, test conditions, test levels and test types, which are essentially hierarchical. It should be noted that classes, which are components in OO paradigm, have two angles. The first is the group of statements (and data) as an extension of structured programming as a way of OOP. The other is constituent of the world as a way of OOA. Test types or test levels may be from the former angle. We should deeply discuss which angle is suitable for test architecture following test requirement analysis and how seamless test requirement analysis models and test architecture models should be.

Relationships are more difficult than statements and components. There are at least two types of relationships. The first is for combinatorial testing. If versions of the OS should be tested combinatorially with versions of Internet Explorer, they have a combinatorial type of relationship. If a load test type should be tested combinatorially with a configuration test type, they also have the combinatorial type of relationship. Another is a sequential type of dependency. As an integration test level should be tested after a unit test level, they have the sequential type of relationship. Other types of relationships than combinatorial and sequential types can be defined if necessary.

In addition, some principles for software design can be applicable such as abstraction, separation of concerns and modularity. Quality characteristics of test suites can indicate and assist good test design such as maintainability of test suites. Notation or formulation can make it easy for engineers to store reusable test assets such as test design patterns and test architecture styles.

### *B. NGT: Notation for Test Architecture Design*

For design of test architecture, notation or a set of concepts is necessary. It should consist of concepts of a group of test cases, hierarchical structure, relationships for combinatorial testing and relationships for sequential dependency. It would be better if it could harmonize the principles, abstraction, separation of concerns, modularity and quality characteristics.

Fig. 4 shows notation or a set of concepts is named NGT, Notation for Generic Testing[2]. NGT consists of three concepts which are viewpoints, hierarchical relationships and interactive relationships. Viewpoints are a concept of a group of test cases. Hierarchical relationships are used for hierarchical structure of viewpoints. Hierarchical relationships mean abstraction (is-a), composition (has-a), cause-effect and object-attribute. Interactive relationships mean necessity for combinatorial testing or are used for sequence of viewpoints. Stereotypes are used for definition of types of viewpoints and relationships. Is-a, has-a, cause-effect, object-attribute, combination and sequence are reperesented as stereotypes.

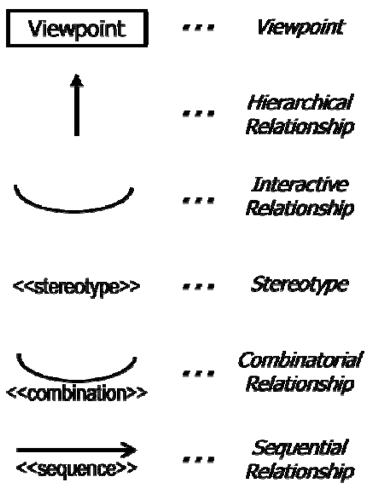

Viewpoint
... Viewpoint
... Hierarchical Relationship
... Interactive Relationship
<<stereotype>>
... Stereotype
<<combination>>
... Combinatorial Relationship
<<sequence>>
... Sequential Relationship

Figure 4. Example of viewpoint diagram and test cases

In Fig.4 the box represents a viewpoint. The directional line with a closed arrowhead represents a hierarchical relationship. The undirectional curved line without an arrowhead represents an interactive relationship. The interactive relationship with a streotype of combination represents a combinatorial relathionship[1]. The interactive relathionship with a streotype of sequence and with open arrowhead represents a sequential relationship.

The bottom viewpoint represents parameters of combinatorial tests or test conditions for test detail design. Test detail design is a phase to extract test cases by test design technique such as simple enumeration, equivalence partitioning, control flow testing and state transition testing. Fig. 5 shows an example of the bottom viewpoint.

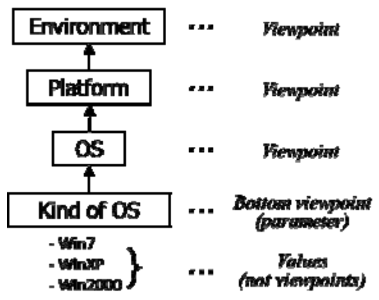
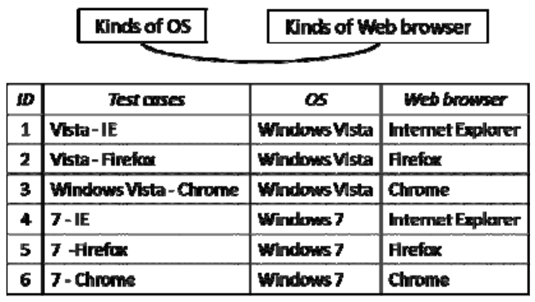


Figure 5. Example of viewpoint diagram and test cases

Fig.6 is an example of a viewpoint diagram and test cases. Each viewpoint indicates parameters and has several values. In this example the viewpoint "Kinds of OS" has two values and the viewpoint "Kinds of Web browser" has three values. One set of combinatorial test cases consists of 6 test cases.

Though the viewpoint diagram looks similar to the classification tree[4], there is a difference between them. The viewpoint diagram tends to be simple in combinatorial test design because it doesn't need to indicate any values and detail combinations. The classification tree, however, tends to be complicated because it needs to indicate each value and each combination in detail. The viewpoint diagram is more suitable than the classification tree for drawing test architecture for combinatorial testing of large-scale and complicated software.



| ID | Test cases | OS | Web browser |
|---|---|---|---|
| 1 | Vista - IE | Windows Vista | Internet Explorer |
| 2 | Vista - Firefox | Windows Vista | Firefox |
| 3 | Windows Vista - Chrome | Windows Vista | Chrome |
| 4 | 7 - IE | Windows 7 | Internet Explorer |
| 5 | 7 -Firefox | Windows 7 | Firefox |
| 6 | 7 - Chrome | Windows 7 | Chrome |

Figure 6. Example of viewpoint diagram and test cases

[1] In this paper all interactive relationships in all diagrams except Fig.4 mean combinatorial relationships.

## IV. Test Architectural Design Patterns for Combinatorial Test Design

There are various patterns in every engineering domain such as constructional design and mechanical design. The patterns are an abstraction of experiences and engineering sense from part of previous successful design. In the software engineering domain there are various patterns such as GoF 23 patterns[5] and analysis patterns[6].

It is impossible to collect patterns exhaustively in any domain where patterns are usually used because they are inductively abstracted from experiences of engineers. Patterns can be structured when several or many patterns are accumulated. If patterns are well-structured enough and a new pattern can be deductively constructed, the well-structured set of patterns will not be called "patterns", but "theories". In the early stages of research such as test architecture design, it is hence important to show examples of patterns though it seems to be ad hoc.

In this paper we show examples of test architecture design patterns below focusing on combinatorial test design using NGT. These patterns are neither exhaustive nor structured because they are inductively abstracted from experiences of selecting parameters and because research of CT architecture design is just in its early stages. Further research on accumulating and structuring patterns is expected.

### 1) *Interaction-Viewpoint Conversion Pattern*

Some kinds of combinations have reasons for being tested combinatorially. For example, the order of installation of software such as service packs of the OS and versions of web browsers can cause a bug which arises from overwriting a shared DLL. When a test designer aims at this bug, he or she cannot only design combinatorial test between service packs of the OS and versions of web browsers but also can design non-combinatorial tests (i.e. tests for just a single parameter) for versions of the shared DLL. His or her selection depends on context such as the clarity of the test objective and the possibility of intentional changes of different versions of a shared DLL. In test architecture design he or she has to be able to convert an interaction into a viewpoint.

Interaction-Viewpoint Conversion is a test architecture design pattern to change an interaction into a viewpoint and to change a viewpoint into an interaction. Fig.7. shows Interaction-Viewpoint Conversion Pattern.

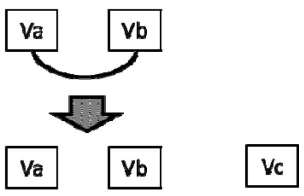


Figure 7. Interaction-Viewpoint Conversion Pattern

For the example of configuration testing for service packs of the OS and several versions of a web browser in Fig.8, Va shows service packs of the OS, Vb shows versions of the web browser and Vc is versions of a shared DLL between the OS and the web browser. Va has 3 values, Vb has 5 values and Vc has 2 values. In this example, some versions of the web browser overwrite the shared DLL of a specific version and the remaining versions overwrite the shared DLL of a different version. A test designer can design 15 test cases for combination of Va (3 values) and Vb (5 values) without this pattern. If he or she aims only at bugs caused by versions of the shared DLL, he or she can apply this pattern and design 10 test cases for non-combination of Va (3 values), Vb (5 values) and Vc (2 values) .

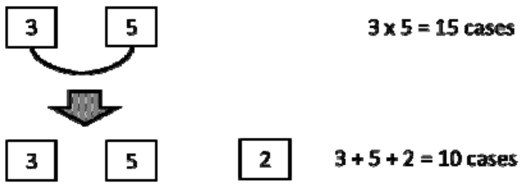


Figure 8. Example of Interaction-Viewpoint Conversion Pattern

### 2) *Interaction Cluster Partitioning Pattern*

There may be a group of complicated viewpoints with a lot of combinations. Each combination doesn't have, however, the same significance. While one type of combination forms some clusters, the other type of combination connects clusters. A cluster can suggest a concern semantically. In test architecture design a test designer should separate concerns and refine a test viewpoint model to increase cohesion and decrease coupling of the model.

Interaction Cluster Partitioning is a test architecture design pattern to divide a complicated clump of combinations into two tight combined clusters and one loose combination between the clusters. Fig.9 shows the Interaction Cluster Partitioning Pattern.

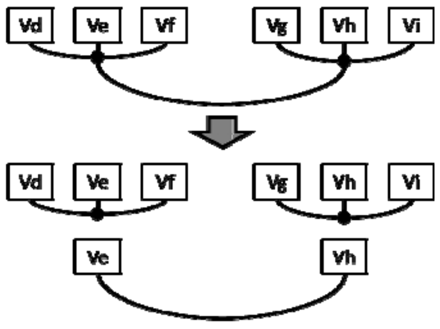


Figure 9. Interaction Cluster Partitioning Pattern

For the example of configuration testing for a client and a server in Fig. 10, Vd is versions of a client OS (3 values), Ve is versions of an e-mail client (4 values) and Vf is versions of a web browser (2 values). Vg is versions of a server OS (2 values), Vh is versions of an e-mail server (3 values) and Vi is versions of an anti-spam server (4 values). In this example, some versions of the e-mail server affect only behaviour of the e-mail client by a bug of SMTP over SSL. A test designer can design 576 test cases for the full combination of Vd, Ve, Vf, Vg, Vh and Vi without this pattern. If he or she aims only at bugs caused by e-mail protocols between the e-mail client and the e-mail server, he or she can apply this patterns and design 64 test cases for two sets of combinations of clusters (Vd-Ve-Vf and Vg-Vh-Vi) and one combination between Ve and Vh. 576 cases and 64 cases can be reduced if CT detail techniques are applied.

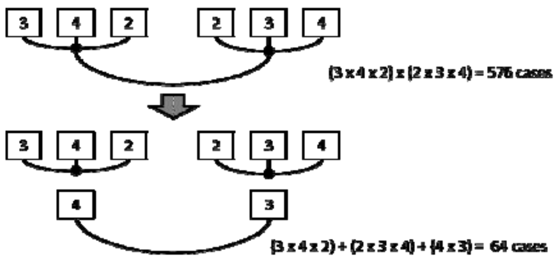

3
4
2
2
3
4
(3 x 4 x 2) x (2 x 3 x 4) = 576 cases
3
4
2
2
3
4
4
3
(3 x 4 x 2) + (2 x 3 x 4) + (4 x 3) = 64 cases

Figure 10. Example of Interaction Cluster Partitioning Pattern

### 3) *Interaction Demotion Pattern*

A test designer may connect semantically different combinations from one abstract viewpoint to other different viewpoints when he or she makes a model with a top-down approach for large-scale and complicated systems. In other words the one viewpoint includes different concerns for the different combinations. He or she should separate the viewpoint into different child viewpoints according to the semantics of the combinations so that he or she can reduce the number of values for each combination.

Interaction Demotion is a test architecture design pattern to divide one viewpoint combined with different viewpoints into its different child viewpoint. Fig.11 shows the Interaction Demotion Pattern.

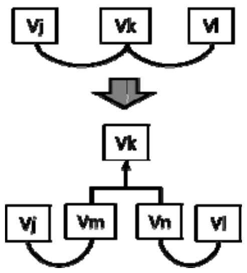


Figure 11. Interaction Demotion Pattern

For the example of testing for a photocopier in Fig. 12, Vj is paper trays (3 values), Vk is paper (6 values) and Vl is duplex mode, that is, one-sided or two-sided (2 values). In this example, paper has two types of values such as size and material. Vm is size (4 values) and Vn is material (2 values). Assuming sizes can affect paper trays and materials can affect duplex mode (via a paper feeding mechanism), while the effect of sizes on duplex mode and the effect of materials on paper trays can be ignored. If a test designer doesn't agree with the assumption, he or she can design 30 test cases for two combinations of Vj-Vk and Vk-Vl without these patterns. If he or she agrees with the assumption, he or she can divide the viewpoint Vk into Vm and Vn, apply this pattern and design 16 test cases for two combinations Vj-Vm and Vn-Vl.

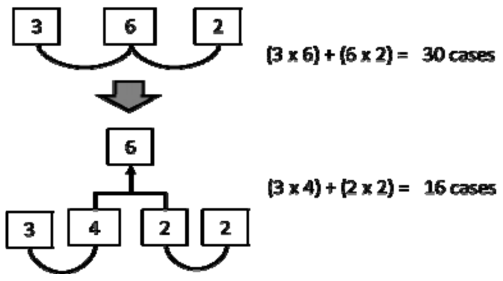


Figure 12. Example of Interaction Demotion Pattern

### 4) *Interaction Necessity Analysis pattern*

Test designers actually tend to increase combinations far more than they are essentially necessary because test designers have a tendency to overestimate risks of omitting combinations far more than they should.

Interaction Necessity Analysis is a test architecture design pattern to review a necessity of interaction and delete the interaction if unnecessary. Fig.13 shows the Interaction Necessity Analysis Pattern.

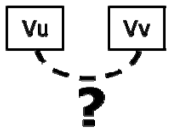


Figure 13. Interaction Necessity Analysis Pattern

## V. CONCLUSION

Software test has recently been a large-scale and complicated artifact as is the software itself. It is necessary to reduce huge combinatorial test cases. This paper focuses on reduction of test parameters and combinations in test architectural design. First we mentioned test architecture design phase in TDLC: Test Development Life Cycle. Second we introduced NGT: Notation for Generic Testing, which is a set of concepts or notation for the design of software test architecture. This paper showed four examples of test architecture design patterns: Interaction-Viewpoint Conversion Pattern, Interaction Cluster Partitioning Pattern, Interaction Demotion Pattern and Interaction Necessity Analysis.

## REFERENCES

[1] OMG, "UML Testing Profile (UTP) Version 1.1 RTF - Beta 1," http://www.omg.org/spec/UTP/1.1/PDF/, June 2011.

[2] Y. Nishi: "Viewpoint based Test Architecture Design," Workshop on Metrics and Standards for Software Testing (MaSST2013), Gaithersburg, Maryland, USA, Jun 2012, CD-ROM(3rd presentation).

[3] IEEE, “IEEE Recommended Practice for Architectural Description of Software-Intensive Systems,” IEEE Std 1471-2000, September 2000.

[4] M. Grochtmann, K. Grimm, “Classification trees for partition testing,” Software Testing, Verification and Reliability, Vol. 3, Issue 2, pp. 63–82, June 1993.

[5] E. Gamma, R. Helm, R. Johnson and J.Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software,” Addison-Wesley, 1994.

[6] M. Fowler: “Analysis Patterns: Reusable Object Models,” Addison-Wesley, 1997.