International Journal of New Technology and Research (IJNTR)
ISSN:2454-4116, Volume-2, Issue-7, July 2016 Pages 59-64

Automatic Generation of Test Cases Using Document Analysis Techniques

Satoshi Masuda, Tohru Matsuodani, Kazuhiko Tsuda

Abstract—**In software maintenance, software testing consumes 55% of the total software maintenance work. The problem is how to reduce the software testing work while still insuring high quality software. Some solutions involve software execution automation tools, outsourcing the testing tasks at lower labor rates. Such solutions still depend upon individual skills in generation of the test cases. In contrast, we focused on generation of test cases rather than the skills and developed a method for the automatic generation of test cases by using our natural language document analysis techniques which use text parsers for extracting and complementing parameter values from documents. We applied the method to Internet banking system maintenance projects and insurance system maintenance projects. In this paper, we discuss our method and techniques for automatic generation of test cases and their use in these industry case studies. Our document analysis tool helped automatically generate 95% of the required test cases from the design documents. The work of creating test cases was reduced by 48% in our case studies.**

Index Terms—**Generation of Test Cases, Document Analysis, Pairwise testing, Software Testing, Test Automation.**

I. INTRODUCTION

Software testing requires high test case coverage [5] as software becomes large and complex [8]. Currently test cases are most often created manually, so test case coverage depends upon each individual skill [10]. In software maintenance, software testing consumes 55% of the total software maintenance work [3],[13]. The problem is how to reduce software testing work while still insuring high quality of software. Some solutions involve software execution automation tools [4],[6], outsourcing the testing tasks at lower labor rates. Such solutions still depend upon each individual skill in testing software. In contrast, we focused on generation of test cases than the skills and developed a method for the automatic generation of test cases by using our natural language document analysis techniques which use text parsers for extracting and complementing parameter values from documents. We applied the method to Internet banking system maintenance projects and insurance system maintenance projects. In this paper, we discuss our method and techniques for automatic generation of test cases and their use in these industry case studies.

The method targets functional testing for Web application systems. The method uses text parsers to identify parameter values for pairwise testing by using our document analysis tool for the design documents with boundary analysis and defects analysis, thus avoiding the dependencies upon individual skills. The document analysis tool uses natural language processing which is a technique for modeling the logic of the documents and testing the analysis [1],[2],[7],[9]. We discuss case studies that demonstrate the method of automatic generation of test cases using document analysis techniques.

Satoshi Masuda, Department of Risk Engineering, University of Tsukuba Faculty of Systems and Information Engineering, Tokyo, Japan.
Tohru Matsuodani, Debug Engineering Research Laboratory, Tokyo, Japan.
Kazuhiko Tsuda, Department of Risk Engineering, University of Tsukuba Faculty of Systems and Information Engineering, Tokyo, Japan.

II. MOTIVATIONS IN AUTOMATED CREATION OF SOFTWARE TESTING CASES

Software testing verifies and validates software as being consistencies with the requirements and design specifications. Effective software testing can improve software quality, but is expensive. As software becomes larger and complex, the costs of software testing can rise exponentially. While both the time for delivery and the work of software maintenance are must be reduced, one key is to improve the efficiency of a software testing. The activities of software testing consist of designing, refining and executing test cases [11]. The activities of testing design and refinement are both important, because they impact the effectiveness and efficiency of software testing. Test cases are created during the test design work, so if too few test cases are created, then the functions of the software are not tested sufficiently and the defects will remain undetected in the software [11].

In this paper, we target functional tests of Web application in software maintenance for the automatic generation of test cases using document analysis technique. Currently, test cases are usually created in three steps. Testing engineers (S1) read the specifications manually, (S2) identify the input parameters and values for the Web screens input parameters, and (S3) apply specialized techniques such as boundary analysis using their expert knowledge. The quality and coverage of the resulting test cases depends upon the skills of the testing engineers, leading to these potential problems [14]:

- Improper understanding of the specifications. Misreading or overlooked documents, basically due to human errors can result in some parameters and values for the test cases being not properly identified.
- Missing parameters and values. In design documents, the types of the parameters are described as character strings, numerical values, dates and so on. Testing engineers create test values of the parameters by using their own knowledge.

- Insufficient combinatorial test cases. It is difficult to manually create combinatorial test cases, again depending upon individual skills in understanding the required test coverage.

These kinds of problems motivate automatic generation of software test cases.

III. CREATING TEST CASES BY USING DOCUMENT ANALYSIS TECHNIQUES

In this section, we discuss about our method to address these problems, reducing the need to depend upon individual skills and using document analysis techniques instead.

A. *Overview*

Figure.1 shows our approach to automatic generation of test cases from the design documents. The document analysis tool reads and analyzes the design documents. Examples of design documents include data specifications for screens, screens design specifications, and event process specifications. By using text parsers to analyze the documents, the analysis tool can output parameters and values with boundary analysis, event conditions, events and expected results. Then our pairwise testing tool uses the parameters and values to create combinatorial test cases [15]. We applied our method to Japanese documents in this paper.

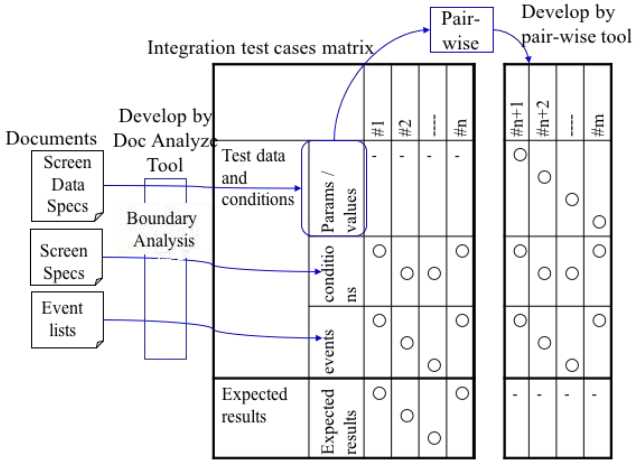

Fig. 1. Automatic creating test cases by using document analysis techniques

The document analysis tool divides its results for the test cases into two parts. The first part is the test data and conditions and the second part is the expected results. The test data and conditions consist of parameters and values, event conditions and events. The expected results are the expected results of execution with the test data and conditions and are described at the bottom of the test cases matrix.

B. *Problems in automatic generation of test cases from documents*

In document-based automatic test case generation, we have two problems: (P1) How to automatically extract the parameter values, such as test conditions and test values, from document set; (P2) How to infer the test conditions and test data that are not explicitly described. The descriptions in the documents have different formats in each project and usually contain natural language descriptions for which information extraction is not easy. So we need flexible information extraction techniques to extract the parameter values from the documents. In addition, the target documents do not explicitly mention all of the required parameter values. This is because such documents are not intended to describe how to test the system. The information that is not described explicitly needs to be complemented or extracted by knowledge about testing and about the systems under test. The extracted information needs to be automatically integrated into the test cases. This also requires special knowledge, such as how to combine parameters and values, which must be derived from the system under test. For these problems, we identify technologies and mechanism that support the automation in each test case generation phase, such as text parser and knowledge for extracting test values.

This figure is an overview of the automatic extraction of parameters and values from the design documents:

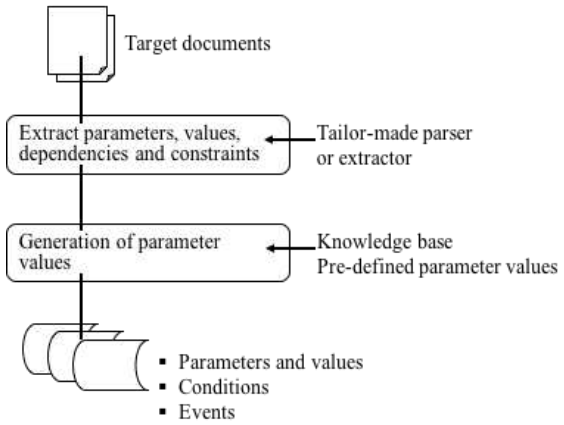

Fig. 2. Overview of automatic extracting parameters and values from design documents

As shown in Figure 2, there are two main phases corresponding to the above two problems facing automatic test case generation. We discuss each phase next two sub sections:

C. *Generation of Parameter Values*

Figure 3 shows the generation of parameter values flow. We need a flexible generation of parameter values to support various kinds of documents and description formats. Recent document analysis and parsing techniques [1],[12] support such flexible information extraction. Document modeling and format checking [1] support the information extraction for various user-defined document-structures and our text parsers combination system [12] makes it easy to create various information extraction parsers at the text description level. For examples, by using the system [12] and combining some formal or natural language parsers, we can flexibly create text parsers that extract the parameters, conditions, values and some dependencies from the text descriptions in target documents.

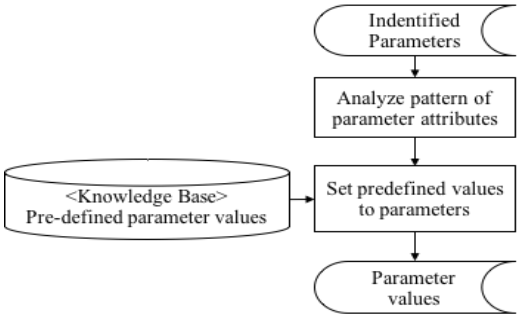

Fig. 3. Generation of parameter values flow

The knowledge of information which is used for identification candidates of parameter values is information that accumulated the results of the boundary analysis pattern from the attributes of the factor in description of specifications. In this paper, we use information which is the results from identify parameter values by patterning the attributes information according with patterns in table I.

TABLE I. KNOWLEDGE PATTERN OF PARAMETER VALUES

Pattern No.	Pattern of parameter values
A1	Character or numeric
A2	Maximum number of characters
A3	Double bytes characters or not
A4	Single byte character or not

TABLE II. EXAMPLE FOR PATTERN OF 4 DIGIT INTEGER PARAMETER VALUES

Pattern No.	Predefined parameter values	Objectives of testing
B1	-1	To test error process for negative values
B2	0	To test error process for zero values
B3	1 and 9999	To test normal process for 4 digit integer
B4	10000	To test error process for 5 digit integer

TABLE III. EXAMPLE FOR PATTERN OF 6 DIGIT CHARACTER PARAMETER VALUES

Pattern No.	Predefined parameter values	Objectives of testing
C1	ABCDEFG	To test 7 digits characters
C2	ABCDEF	To test 6 digits characters
C3	ABCDE	To test 5 digits characters
C4	アイウエオ	To test 5 digit double bytes Katakana characters
C5	ｱｲｳｴｵ	To test 5 digit single byte Katakana characters
C6	123	To test numeric

Document analysis tool identify parameter values by suing boundary analysis and apply predefined values from the results of analysis about the attributes pattern, such as a character string, a numerical value and length, and a digit number, and so on. This enables not to depend on individual skill, the parameter values can be identified a level of coverage and used for test cases. Comparing create test cases by manually, this way makes test cases have better quality and more efficient in creation of test cases.

For examples about predefined values in the case of the attribute of the integer of 4 digits, by an experienced person's knowledge, parameter values are created by the patterns showed in table II. The pattern number from B1 to B4 come from the results of boundary analysis and B5 comes from the knowledge of problem information. The document analysis tool creates the predefined values according with the attributes of values.

Another example about predefined vales in the case of the attribute of the character string of 6 digits, by using the knowledge of experienced person the predefined parameter values are created in a similar manner as table III. The pattern number from C1 to C4 come from the results of boundary analysis and C5 and C6 comes from the knowledge of problem information.

D. Problems in automatic generation of test cases from documents

In the last step, the pairwise testing tool creates combinatorial test cases from the identified parameters and values. So we can create the combinatorial pairwise test cases not depending upon individual skill. The test cases have a level of quality as high as pairwise combination, and the work of generation test cases are reduced by using the tool.

Each technique in our method is known techniques, such as text parser, boundary analysis, pairwise testing and so on. Our method is, however, unique for combining of them to create test cases automatically from design documents. Our method also solves the problems which are miss-reading of the documents, lacks of identification of parameter values, and insufficient of a combination test cases. Comparing manual test cases as ideal test case [12], our document analysis tool helped automatically generate 95% of the required test cases from the design documents.

IV. CASE STUDIES FOR APPLYING THE METHOD

We applied our method to testing for Web applications in internet banking system and insurance system. The case studies are functional testing by inputting test data from Web screens. The Web application documents are Web screens parameter specifications which describe attributes of parameter on Web screens input data, Web screens design documents which describe screens layout and precondition of events, and an event lists which describe the post processes of the events by clicking the buttons. In order to compare activities and work between manual way of creating test cases and our method way of automatic generation of test cases, each activity of creating testing cases are listed in Table VI. The activities list shows activity number, name of activities, breakdown activities and comparing manual way and our method.

We applied our method to the following four case studies:

- Case 1: The generation test cases from an Internet Banking system project.
- Case 2: The generation test cases from another Internet Banking system project.
- Case 3: The generation test cases from Insurance system the input data of a batch job.
- Case 4: The generation test cases from Insurance system the screens of Web application

Table V shows the results of work. Each of case studies is compared between manual way and our method. The work in the generation test cases by our method was reduced from 23% to 48%. So our method improved work of generation test cases about twice as much as manual way.

TABLE IV. CREATING TEST CASES ACTIVITIES COMPARISON

Act No.	Activities	Break down activities	Manual way	Our method
1-1	Create test data and variation	Analyze document	Analyze manually	Analyze by tools
1-2		Create test data	Identify manually	Identify by tools
1-3		Create parameter values	Create manually	Create by tools
2-1	Identify pre-conditions	Analyze document	Analyze manually	Analyze by tools
2-2		Identify pre-conditions	Identify manually	Identify by tools
3-1	Identify event conditions	Analyze document	Analyze manually	Analyze by tools
3-2		Identify event conditions	Identify manually	Identify by tools
4-1	Create test cases	Create test cases which verify conditions	Create manually	Create test cases by tools.
4-2		Create combinatorial test cases.	Create manually	Create by pairwise tools
5	Review test cases	Review test cases	Review manually	Review manually

TABLE V. WORKLOAD COMPARISON OF TEST CASE GENERATION

	Case1		Case2		Case3[a]		Case4 [a]	
No.	*M(h)*[c]	*O(h)*[c]	*M(h)*	*O(h)*	*M(h)*	*O(h)*	*M(h)*	*O(h)*
1-1	5	1	5	1	2	1	1	1
1-2	12	1	5	1	3	1	2	1
1-3	10	4	10	3	4	2	2	1
2-1	5	1	5	1	2	1	1	1
2-2	13	5	15	4	6	3	3	2
3-1	8	1	5	1	-[b]	-[b]	1	1
3-2	15	5	15	4	-[b]	-[b]	3	2
4-1	62	42	60	50	10	8	4	4
4-2	38	20	40	30	10	7	4	3
5	15	15	10	10	3	3	1	1
Tot.	183	95	170	105	40	26	22	17
Reduction of work		48%		38%		35%		23%

[a] About the workload of the case 3 and 4, the work of manual method is from interviews and the work of our method are calculated out on paper

[b] The case 3 did not have event description of a button, a link, etc. for batch input data.

[c] M(h): Manual results(hours), O(h): Our method results(hours)

V. OBSERVATIONS AND LESSONS LEARNED

We devised a method of automatic generation of test cases by using document analysis technique and applied the method to several case studies. The work for the generation of test cases was reduced by up to 48%. We demonstrated that our new method improved the effectiveness creating test cases and also studied how we could apply our method to actual cases. If we can automate the review of the test cases, then the improvement will be larger. In the actual case studies, all of the design documents were not formalized according to the document standards, because the participants had been allowed to freely describe their design specifications. Notwithstanding, our text parser was able to read their document and create up to 95% of the required test cases for satisfactory testing.

VI. RELATED WORKS

Natural language processing and checking consistency are essential in requirement engineering. Yan2015 present formal consistency checking over specifications in natural languages [16]. The paper present: a requirement consistency maintenance framework to produce consistent representations. The first part is the automatic translation from natural languages describing functionalities to formal logic with an abstraction of time. It extends pure syntactic parsing by adding semantic reasoning and the support of partitioning input and output variables. The second part is the use of synthesis techniques to examine if the requirements are consistent in terms of realizability [16]. This paper presents the differences from Yan2015 as follows, abstraction logic by transforming propositional logic not only time constraints, using input data patterns to find logical inconsistency and semantic role labeling.

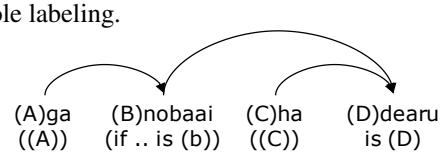

Fig. 4. An example condition logic dependency [17] in Japanese.

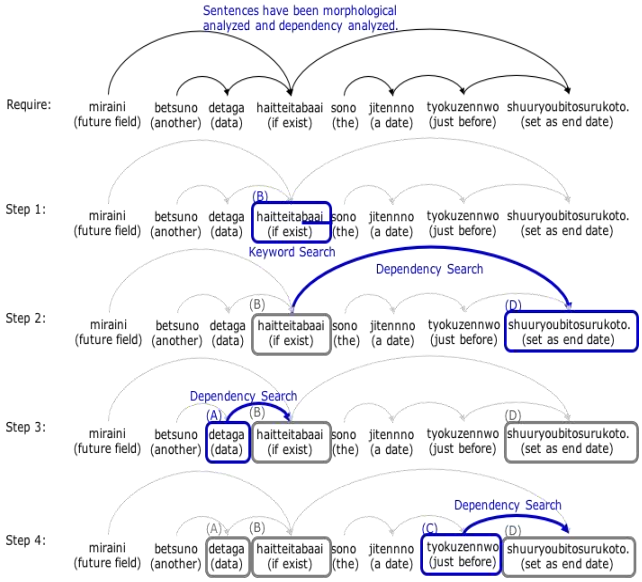

Fig. 5. Steps to retrieve conditions and actions from sentences [17] in Japanese.

SAT solver is a solution for checking logical constraint [26]. SAT solves about validity and consistency. The validity and consistency are really two ways of looking at the same thing and each may be described in terms of syntax or semantics [28]. Combinatorial testing is the solution for data patterns. Combinatorial Testing (CT) can detect failures triggered by interactions of parameters in the Software Under Test (SUT) with a covering array test suite generated by some sampling mechanisms [27]. There is a method of creating testing pattern for Pair-wise method by using knowledge of parameter values. The method uses knowledge base for identifying pair-wise parameter values by using document analysis to specification documents, boundary analysis and defects analysis [14].

Figure 4 and 5 shows the logic to which is "If (A) is (B), (C) is (D)." from a sentence by natural-language processing [17], but the condition and actions are not identified. There is also linguistic work for use cases [22] and test case generation [25]. There is Sneed2007 for the study which makes test case from a described required specification by a natural language [23]. The approach which defines classification of how to interpret the specification of the etc. which "is description of output" which "is description of input" as a key word and applies that. There aren't a morphological analysis and syntactic analysis/analyses using analysis technology of natural-language processing, and this can think there are few generalities. There are other related works. Bos2007 offers the way to change from a natural sentence to a logical expression and tool BOXER in English [18].

IEEE830 recommend practice for software requirements specifications [19], [20]. The standard describes consideration for producing a good software requirements specification, parts of them and provide templates. Kim2008 [21] presented measurement of level of quality control activities in software development. The ambiguity definition was described in the paper. When we target natural language requirements, these two papers are very effective at the point of view, how requirements should be described and how measure them. Uetsuki2013 presented an efficient software testing method by decision table verification [24]. Figure 6 shows whole process of decision table logic verification. That was to verify logics between documents and source codes by comparing each decision tables which were extracting from documents and codes. They targeted the document was described formal language.

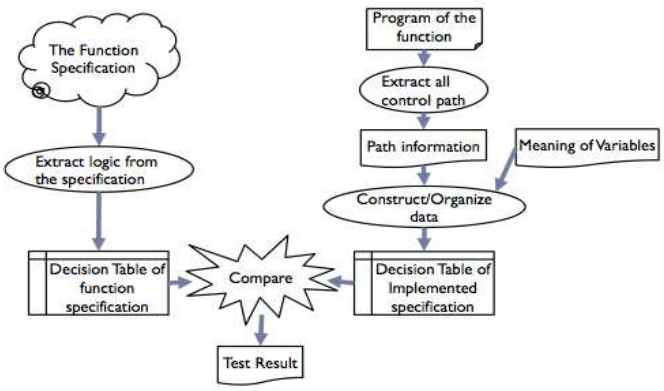

Fig. 6. Whole process of decision table logic verification in [25]

VII. CONCLUSIONS

In this paper, we discussed a method of automatic generation testing cases by using document analysis techniques. We also discussed four case studies which demonstrated the efficiency of our approach in creating high-coverage test cases by using the document analysis tool. The method targets functional testing of the interaction screens for Web application systems. The method uses text parsers that identify input parameters and their acceptable values by using document analysis on the design documents, thus avoiding dependencies upon individual skills.

There are still some problems with how we extract the information from the design documents. Future work includes applying on other actual cases in order to get learn how we can configure the text parsers for natural languages. We are also studying and developing document models of the design documents to help formalize the documents.

REFERENCES

[1] T. Nakamura. “Enabling analysis and measurement of conventional software development documents using project-specific formalism”, 6th International Conference on Software Process and Product Measurement, 2011, pp.48-54.
[2] T. Nakamura, H. Takeuchi, “Document quality verification tool”, ProVision No.69;2011, pp.78-79. (In Japanese)
[3] IPA. METI, “Report of industry actual survey for embedded software in 2009”, 2009 (in Japanese)
[4] C. Cadar, “EXE: automatically generating inputs of death.”, ACM Conference on Computer and Communications Security, Vol. 12, No. 2, Article 10;2006
[5] D. Cohen, “The AETG system: an approach to testing based on combinatorial design” IEEE Transactions on Software Engineering, Vol. 23, No. 7,1997, pp.437-444
[6] M. Sharma, “Automatic generation of test suites from decision table - theory and implementation”, Fifth International Conference on Software Engineering Advances, 2010, pp.459-464.
[7] A. Sinha, A. Paradkar, H. Takeuchi, T. Nakamura, “Extending automatic analysis of natural language use cases to other languages”, in Proceedings of the 18th IEEE International Requirements Engineering Conference, 2010, pp. 364-369.
[8] B. Curtis, H. Kransner, and N. Iscoe, “A field study of the software design process for large scale systems”, Communications of the ACM, Vol. 31, No. 11; 1988, pp.1268-1287
[9] H. Takeuchi, L V Subramaniam, T. Nasukawa, S. Roy, S. Balakrishan. “A conversation-mining system for gathering insights to improve agent productivity”, IEEE Joint Conference on E-Commerce Technology (CEC’07), 2007, pp.465-468
[10] D. Cohen,“The combinatorial design approach to automatic test generation”, IEEE Software Engineering; 1996, pp.83-88
[11] K. Uetsuki, “Research about a design technique of software testing by using decision table”, 2013, pp.1-6 (in Japanese)
[12] F. Iwama, T. Nakamura, H. Takeuchi, “Constructing Parser for Industrial Software Specifications Containing Formal and Natural Language Description” in ICSE, 2012, pp1012-1020
[13] D. Reifer, J. Allen, B. Fersch, B. Hitchings, J. Judy, “Total cost of software maintenance workshop” in USC CSSE, 2010, p.11
[14] S. Masuda, T. Matsuodani, K. Tsuda,” A method of creating testing pattern for pair-wise method by using knowledge of parameter values”, 17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems(KES), 2013
[15] I. Segall, R. Tzoref-Brill, E. Farchi,“Using Binary Decision Diagrams for Combinatorial Test Design”, ISSTA '11 Proceedings of the 2011 International Symposium on Software Testing and Analysis, 2011, pp 254-264
[16] R. Yan, C.-H. Cheng, and Y. Chai,"Formal consistency checking over specifications in natural languages", in EDA ConsortiumProceedings of the 2015 Design, Automation \& Test in Europe Conference \& Exhibition,2015,pp.1677--1682
[17] S. Masuda, F. Iwama, N. Hosokawa, T. Matsuodani, and K.Tsuda, "Semantic analysis technique of logics retrieval for software testing from specification documents", in IEEESoftware Testing, Verification

and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on,2015,pp.1--6
[18] J. Bos,"Wide-coverage semantic analysis with boxer", in Association for Computational LinguisticsProceedings of the 2008 Conference on Semantics in Text Processing,2008,pp.277--286
[19] I. C. S. S. E. S. Committee, and I.-S. S. Board,"IEEE Recommended Practice for Software Requirements Specifications", in Institute of Electrical and Electronics Engineers,1998
[20] ISO/IEC/IEEE,"Software and systems engineering — Software testing — Part 4:Test techniques", ISO/IEC/IEEE JTC 1/SC 7,2015,pp.70--72
[21] C. Kim, S.-M. Kim, and K.-W. Song, "Measurement of Level of Quality Control Activities in Software Development [Quality Control Scorecards]", in IEEEConvergence and Hybrid Information Technology, 2008. ICHIT'08. International Conference on,2008,pp.763--770
[22] A. Sinha, A. Paradkar, P. Kumanan, and B. Boguraev,"A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases", in IEEEDependable Systems \& Networks, 2009. DSN'09. IEEE/IFIP International Conference on,2009,pp.327--336
[23] H. M. Sneed,"Testing against natural language requirements", in IEEEQuality Software, 2007. QSIC'07. Seventh International Conference on,2007,pp.380--387
[24] K. Uetsuki, T. Matsuodani, and K. Tsuda,"An efficient software testing method by decision table verification", in International Journal of Computer Applications in Technology,2013,pp.54--64
[25] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal,"Automatic generation of system test cases from use case specifications", in ACMProceedings of the 2015 International Symposium on Software Testing and Analysis, 2015,pp.385--396
[26] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith,"Constraint models for the covering test problem", in Constraints, 2006,pp.199—219
[27] C. Nie, and H. Leung,"A survey of combinatorial testing", in ACM Computing Surveys (CSUR), 2011,pp.11
[28] A. Biere, M. Heule, and H. van Maaren,"Handbook of satisfiability", in ,2009

Satoshi Masuda received his B.S. in Mechanical Engineering from Saitama University in 1991. His research interests include software testing, natural language processing, information retrieval and algorithm. He is a senior member of The Information Processing Society Japan and a member of IEEE Computer Society.

Tohru Matsuodani received his Ph.D. from Tsukuba University in 2005. He has been chief executive officer of the Debug Engineering Laboratory. He is a member of The Information Processing Society Japan and the IEEE computer society.

Kazuhiko Tsuda has been working as a Professor at the Graduate School of Business Sciences, University of Tsukuba, Tokyo, Japan since 1998. He received his BS and PhD in Engineering from Tokushima University in 1986 and 1994, respectively. His research interests include natural language processing, database, information retrieval, algorithm and human-computer interaction. He is a member of IEEE Computer Society, The Information Processing Society of Japan and The Institute of Electronics.