2016 IEEE International Conference on Software Quality, Reliability and Security

# Applying Combinatorial Testing in Industrial Settings

Xuelin Li, Ruizhi Gao, W. Eric Wong
Department of Computer Science
The University of Texas at Dallas, USA
{xxl131630, gxr116020, ewong}@utdallas.edu

Chunhui Yang, Dong Li
The Fifth Electronics Research Institute of Ministry of Industry and Information Technology (CEPREI), China
{yangch, lidong}@ceprei.com

*Abstract* — **Combinatorial testing (CT) is a black-box-based technique to generate a small number of test cases with a focus on covering various interactions among input parameters of a software system. How effective is CT in practice? What are the challenges and issues presented by the process of applying CT? How can these challenges and issues be overcome to improve the application of CT? Although CT has attracted attention from both academia and industry, these questions have not been addressed or researched in depth based on an empirical study in industrial settings. From January 2016 to February 2016, we have worked with CEPREI, an authoritative software testing company in China, on testing three real-life software systems using CT. Throughout the process, we have generated 601 test cases and found 33 bugs in total. The results demonstrate that CT not only detects more bugs but also requires less time in designing test cases for a specific system comparing to traditional function coverage-based testing. This paper will present our empirical study thoroughly and give an insightful analysis to investigate the advantages as well as the possible challenges of applying CT.**

***Keywords—Combinatorial Testing, real-life industrial projects, fault detection***

## I. Introduction

Software has become fundamental to our society and our everyday lives. Regardless of age, gender, occupation, or nationality, each one of us depends on software, either directly or indirectly. Yet the disappointing truth is that software is far from defect-free and very large sums of money are spent each year to fix and maintain defective software. According to a National Institute of Standards and Technology report [18], software bugs cost the U.S. economy an estimated $59.5 billion annually.

Another concern is that even though we understand the importance of delivering software systems with high quality, due to the lack of universal tools, software testing in industry still relies greatly on testers' experience and understanding of the system under test (SUT). As a result, it becomes significantly difficult to standardize the software testing process and to develop a universal testing framework for software systems.

In order to deal with the concern, various standards, such as DO-178b, have been applied to ensure that testing of a specific software system has been sufficiently performed. However, two shortcomings are evident: first, it is unknown whether the predefined standards are applicable for a specific software system; second, instead of illustrating how to perform software testing, the standards provide only to what degree a software system should be tested.

CT could be a good candidate to tackle these problems [2,3,4,6,9,19]. First, CT is claimed to be a cost-effective and easy-to-learn technique for generating high quality test cases [2,3,7,17]. The particularity of CT is that it can generate a very small number of tests to cover certain specified combinations between input parameters of a software system. Also, light-weight tools such as ACTS [1] are applicable for most of software systems, which makes CT a practical approach for companies to reduce the cost of testing and improve efficiency.

Despite the potential advantages, there are still comparatively little industrial studies on CT, particularly with focus on investigating how effective CT could be in practice, what the challenges of applying CT are, and how to tackle the challenges, which become the major incentives and research topics of this paper.

To answer these questions, in cooperation with CEPREI (an authoritative software testing company in China), we apply CT to three real-life industrial software systems. Our result shows that by using CT, not only the cost of testing significantly decreases, but also more bugs are detected than using traditional function coverage-based testing. A detailed presentation of our study will be given in Section IV. Analysis of the application of CT to the three pilot projects as well as the positive and negative lessons learned from the process are presented in Section V.

The remainder of the paper is organized as follows: in Section II, we describe the basic concepts of CT. What follows in Section III is the description of three pilot projects used in our study. The test generation and corresponding analyses are provided in Section IV. In Section V, we focus on the lessons

978-1-5090-4127-5/16 $31.00 © 2016 IEEE
DOI 10.1109/QRS.2016.16

learned from the application of CT to the pilot projects, followed by our conclusion and future work in Section VI.

## II. Combinatorial Testing

An interesting phenomenon which occurs frequently with large software systems draws attentions from both academy and industry: after successful deployment for a long period of time, users suddenly encounter unexpected failures and a new set of bugs. Such failures are often referred to as *interaction failures*, which are caused by the combinations of two or more input parameters [7,11,12,14]. Even though the systems have been thoroughly tested and extensively used, there still exist combinations which could cause the system to reach an incorrect state.

In order to resolve this concern, we could perform exhaustive testing with respect to all input parameters. For example, assume we have a program *P* with *k* input parameters. With respect to the *i*th parameter, it has $v_i$ possible values, where $1 \le i \le k$ . Therefore, a total number of $\prod_{i=1}^{k} v_i$ test cases are needed to completely test all the combinations. Undoubtedly, the cost of performing these tests is high which makes such testing inapplicable for most of software systems.

According to previous publications [5,13,16], a large percentage of software failures are often caused by a limited number of input parameters. By analyzing the bug reports of Mozilla browser, Kuhn et al found that more than 70% failures are caused by the interaction of two parameters. The number grows to 90% when the interactions of no more than three parameters are considered. As a result, we can generate a test set such that every combination of possible values of *t* input parameters is covered at least once. Normally, *t* is less than six to avoid too many generated test cases. The test generation technique is called *t*-way CT [2,13,15].

Below we include a simple example of how to apply 2-way CT. Suppose the SUT accepts four parameters, each of which represents the status of a signal light. Every parameter has two possible values: "0" stands for the light is "off" while "1" means "on".

- Parameter A: signal light A
- Parameter B: signal light B
- Parameter C: signal light C
- Parameter D: signal light D

By using exhaustive testing, we need $2 \times 2 \times 2 \times 2 = 16$ test cases. However, the approach cannot be applied to real-life industrial projects, since the number of input parameters is great, not to mention that each parameter can have multiple possible values.

Instead, one acceptable trade-off is to design test cases which cover all the combinations of possible values of 2 input parameters (2-way CT). Table 1 below contains all the possible combinations:

Table 1. All possible combinations between two input parameters

| | | | |
|---|---|---|---|
| A = 0, B = 0 | A = 0, B = 1 | A = 0, C = 0 | A = 0, C = 1 |
| A = 0, D = 0 | A = 0, D = 1 | A = 1, B = 0 | A = 1, B = 1 |
| A = 1, C = 0 | A = 1, C = 1 | A = 1, D = 0 | A = 1, D = 1 |
| B = 0, C = 0 | B = 0, C = 1 | B = 0, D = 0 | B = 0, D = 1 |
| B = 1, C = 0 | B = 1, C = 1 | B = 1, D = 0 | B = 1, D = 1 |
| C = 0, D = 0 | C = 0, D = 1 | C = 1, D = 0 | C = 1, D = 1 |

In order to cover the 24 combinations, by using IPOG algorithm [10], we need only six test cases:

- A = 1, B = 1, C = 0, D = 0
- A = 1, B = 0, C = 1, D = 1
- A = 0, B = 1, C = 1, D = 0
- A = 0, B = 0, C = 0, D = 1
- A = 0, B = 1, C = 0, D = 1
- A = 1, B = 0, C = 0, D = 0

Among various algorithms for Combinatorial Testing, we employ the IPOG algorithm for the rest of the paper. With respect to the tool for test generation, the ACTS [1] developed by National Institute of Standards and Technology is used. The tool has the ability to generate tests for 2-way through 6-way interactions, with a user-friendly GUI and a command line version suitable for use in scripts or system calls from another tool. It has been proved to be effective in the application of CT to several industrial projects [7,8].

## III. Project Description

### A. *Dashboard of Subway Control System (DSCS)*

DSCS is an embedded software system which controls the display of real-time data related to the status of a locomotive. The hardware platform for DSCS is shown in Figure 1.

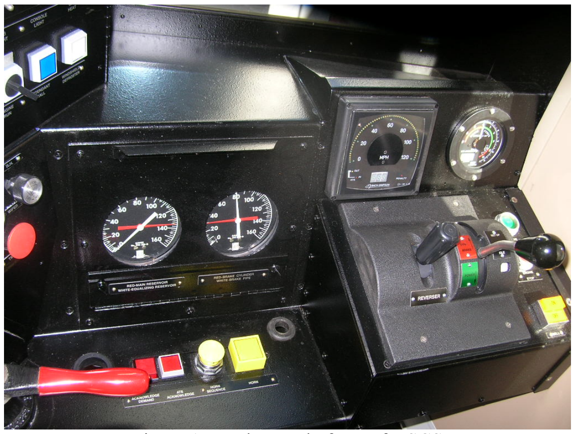


Figure 1. Hardware platform of DSCS

The main interface of the system is shown in Figure 2. The most important status information of the locomotive is demonstrated on this interface, such as equalizing reservoir pressure, capacity of air flow, breaking cylinder pressure, and so on.

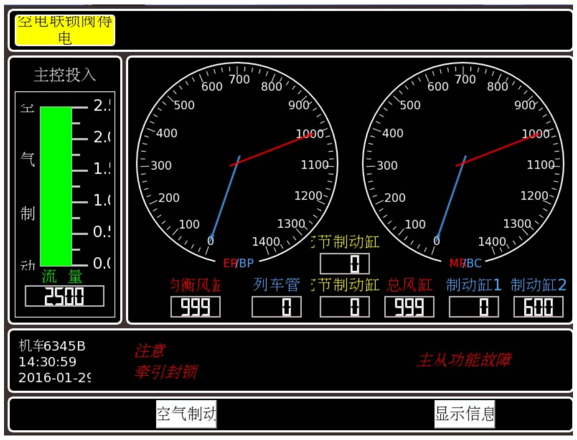


Figure 2. Main interface of DSCS

More detailed information regarding the locomotive is displayed in the secondary interface as in Figure 3. It contains additional information such as status of each valve, status of the breaking controller, communication status via BCU module, and so on.

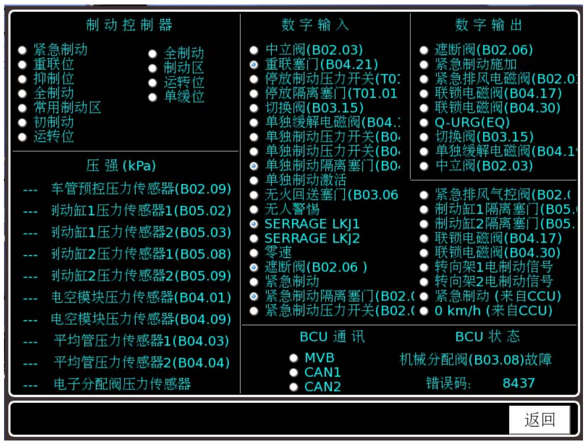


Figure 3. Additional status information provided by DSCS

The inputs of the system are categorized as four distinctive data packages, denoted as DP I, II, III, and IV. Each of the packages are comprised of 84 hexadecimal numbers. As a result, test case for the system can be represented as a string of 84 84 hexadecimal numbers. One sample test case is as follows:

$t$ = {1616FF80002103800 1F4000078E703E7…7EE9FF}

The first 14 numbers of each input string are used to represent in which data package it belongs. With respect to the above test case, numbers in red, {1616FF80002103}, indicate that the test case is designed for DP III. The last six numbers, {7EE9FF} marked in orange, are error correcting codes which ensure the correctness of each input string. All other numbers represent specific values of data sent from the central control unit of a locomotive.

For example, as illustrated in Table 2, with respect to the eighth byte of DP III, each bit within the byte represents whether a specific system failure occurs. In test case $t$, the eighth byte marked in green, “80”, represents that currently the system encounters Error I, since $(80)_{16} = (1000\ 0000)_2$.

Table 2. The eighth byte of DP III[1]

| Eighth byte of DP III | | | | | | | |
|---|---|---|---|---|---|---|---|
| Error I | Error II | Error III | Error IV | Error V | Error VI | Error VII | Error VIII |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

On the other hand, some parameters may need more than one bit. Take the air pressure in breaking cylinders as an example, which is represented by the ninth and tenth byte of DP III. The range of the value is from 0 to 999 kPa. As a result, two bytes (16 bits) are employed to represent a specific air pressure. As in test case $t$, 500 kPa is denoted as “01F4” marked in purple, since $(500)_{10} = (0000\ 0001\ 1111\ 0100)_2 = (01F4)_{16}$.

## B. *NeoKylin Operating System (NeoKylin)*

NeoKylin, which is a graphical operating system, is developed by China Standard Software and National University of Defense Technology since December 2010. The system is designed for government offices, national defense, energy and other sectors of the Chinese government. A screenshot of the system is displayed in Figure 4.

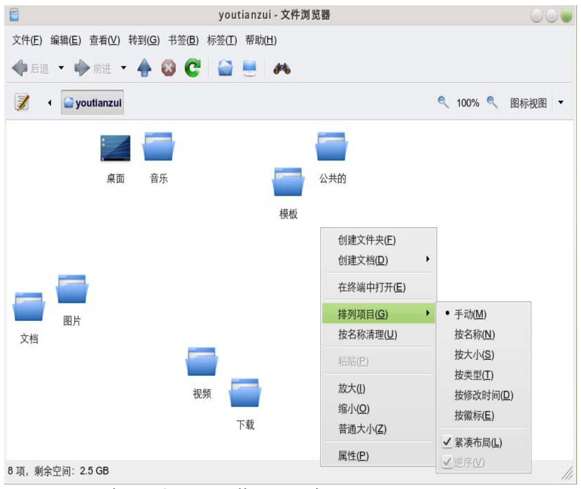


Figure 4. NeoKylin operating system

Based on the functionalities provided by the system, our test generation focuses on the following aspects:

- Hardware configuration testing, which examines whether the configuration of CPU, hard drives, graphic cards, etc., can be correctly displayed.
- Interface testing, including testing for USB interface, Audio/Video interface, etc.

[1] For credential purposes, technical details of the system are not presented.

- Basic Function testing, including testing for operating system management, resource management, installation and uninstallation of specific applications, etc.
- Performance testing.
- Other non-functional features of the system.

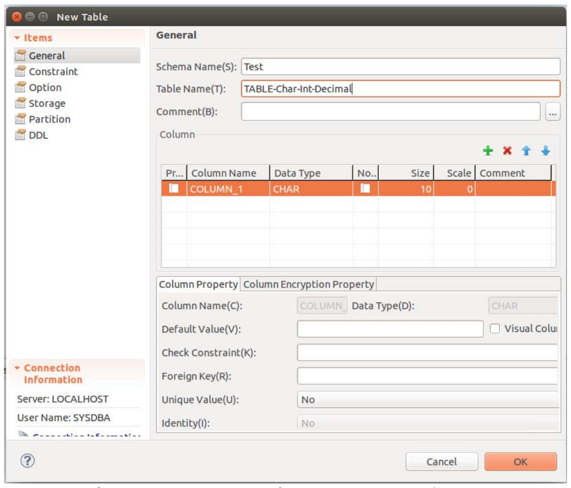
## C. DaMeng Database Management System (DM)

DM is a large scale management system with strong capabilities of analyzing and processing innumerous data. Nowadays, China's ministries and departments are highly dependent on the database due to its cutting-edged advantages of security and excellent performances. Thirty years of cooperation with high-end and new technology industries enable DM to be applied in numerous fields and receive large-scaled recognition.

Since 2012, DM has been replacing international counterparts as the main service supplier for many entities such as Ministry of Health and provincial library in Hubei. The system shoulders the responsibilities to rejuvenate database industry and plays a significant role in information security, government working, business security and national defense industries. A screenshot of the system is shown Figure 5.

Based on the functionalities provided by the system, our test generation focuses on the following aspects:

- Functional testing, including installation, role management, user management, table management, index management.
- Reliability testing, including online backup, off-line backup, incremental backup.
- Performance testing, including response time, database saving time, efficiency in batch upload/download.
- Other non-functional features of the system.



General

Schema Name(S): Test

Table Name(T): TABLE-Char-Int-Decimal

Comment(B):

| Pr... | Column Name | Data Type | No.. | Size | Scale | Comment |
|---|---|---|---|---|---|---|
| ☐ | COLUMN_1 | CHAR | ☐ | 10 | 0 | |

Column Name(C): COLUMN_ Data Type(D): CHAR

Default Value(V): ☐ Visual Colu

Check Constraint(K):

Foreign Key(R):

Unique Value(U): No

Identity(I): No

Server: LOCALHOST

User Name: SYSDBA

Figure 5. DaMeng Database Management System

## IV. Test Generation and Results

To validate the effectiveness of CT, we compare CT with traditional function coverage-based testing (FT) which focuses on the verification of whether each function provided by the system has been correctly implemented with respect to their fault detection strength and number of test cases needed. Our results regarding each pilot projects are listed in the following sub-sections, respectively.

### A. DSCS

Since DSCS contains 199 input parameters in total, the total number of test cases generated using 2-way CT is more than 1,000. If we only consider one specific data package, for example DP IV with 50 input parameters, ACTS generates more than 400 test cases for 2-way CT. Even though the test cases can be executed automatically, the verification of outputs have to be performed manually, which would consume tremendous amount of time. As a result, it is mandatory to limit the number of test cases.

Our solution to this challenge is: instead of identifying each input parameter as an independent parameter, several parameters within the same byte are combined as one input parameter. With respect to the eighth byte of DP III in Table 2, the whole byte will be recognized as one input parameter, "Eighth Byte". It contains nine possible input values: eight of them represent the presence of a particular error while the ninth stands for the absence of error, as displayed in Table 3.

Table 4. "Eighth Byte" as input parameter

| Eighth Byte | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Error I | Error II | Error III | Error IV | Error V | Error VI | Error VII | Error VIII | Input value |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 40 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 20 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 08 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |

In this way, the number of test cases can be limited to an acceptable scale. In Table 5, we have listed the number of test cases generated by CT/FT, number of bugs found by CT/FT, and whether CT found all the bugs discovered by FT. The following observations can be seen from Table 5.

Table 5. Results for DSCS

| | | Number of test cases | Number of bugs found | CT find all FT bugs? |
|---|---|---|---|---|
| DP I | FT | 98 | 2 | Yes |
| | CT | 49 | 6 | |
| DP II | FT | 102 | 1 | Yes |
| | CT | 77 | 5 | |
| DP III | FT | 116 | 2 | Missed 1 |
| | CT | 80 | 7 | |
| DP IV | FT | 122 | 2 | Yes |
| | CT | 90 | 4 | |

#### 1) 2-way CT requires fewer test cases than FT

Regarding the four data packages, CT generates fewer test cases than FT. For example, with respect to DP I, FT generates 98 test cases while 2-way CT only requires 49 test cases to be executed in order to thoroughly examine the functionalities related to DP III. The same applies to other three data packages.

*2) 2-way CT detects more bugs than FT*

With respect to all four data packages, 2-way CT can always find more bugs than FT. For example, when testing DP II, 2-way CT detected five unique bugs while FT only found one.

*3) 2-way CT detects most of bugs found by FT*

Referring to the table, CT detected all the bugs that FT found with only one exception: with respect to DP III, though 2-way CT detected seven bugs and FT only detected two bugs, 2-way CT missed one bug found by FT. However, for most of the cases, 2-way CT shows significant better performance than FT in detecting bugs.

## *B. NeoKylin*

The most significant obstacle in applying CT to test NeoKylin is the difficulty in parameterization of inputs. Sample inputs with respect to the resource management module are as follows:

- Arrange desktop icons by name, file size, file type, modified date, or manually adjust the order;
- Drag an icon;
- Choose display size (33%, 100%, 150%, or 400%);
- Choose display view (icon, list, or compact).

As a result, we elicit different types of operations performed by users as the input parameters. Possible values of the parameters will be the different options provided by the system. For example, "arrangement of desktop icons" is one input parameter and the five options "by name", "by file size", "by file type", "by modified date", and "manually adjust" become the possible values of the input parameter. In this way, by combining operations, we establish different operational profiles in order to thoroughly test the NeoKylin operating system.

Similar to Table 5, Table 6 manifests the results of applying 2-way CT and FT, respectively, to NeoKylin Operating System. Though 2-way CT requires 25 more test cases, it detected 11 bugs compared to FT, which only found four.

In addition, 2-way CT only missed one single bug discovered by FT, and it is highly possible that this bug can be detected by CT with higher strength, e.g., 3-way CT.

Table 6. Results for NeoKylin

| | | Number of test cases | Number of bugs found | CT find all FT bugs? |
|---|---|---|---|---|
| NeoKylin | FT | 147 | 4 | Missed 1 |
| | CT | 172 | 11 | |

## *C. DaMeng Database Management System*

Since DaMeng Databases Management System has been developed and widely used for more than ten years, it is a high quality system. As a result, both teams haven't detected any bugs in the version provided by DaMeng Database Corporation. However, during the testing process, we find that the cost of designing test cases significantly decreases. For example, by using FT, it usually takes two hours for a tester to design the test cases for one specific functionality provided by the database system. In contrast, by using CT and ACTS tools provided by NIST, it only requires less than half an hour to complete the same task. As a result, the efficiency of testing is significantly improved.

## V. Lessons Learned

Compared to using traditional functional coverage-based testing, it has been demonstrated that the application of CT not only helps engineers detect more bugs but also lowers the testing cost, which proves that CT can be an effective and affordable technique for testing embedded software systems (DSCS), operating systems (NeoKylin), and applications (DaMeng Database Management System).

After the success of the three pilot projects described above, an interesting discussion arises among engineers at CEPREI. It includes not only the experience and lessons learned from the projects but also the scope of applying CT.

## *A. Single parameter or multiple parameters?*

Though the aim of CT is to design test sets that have the ability to detect those failures triggered by the combination of multiple input parameters, it can also help us find those triggered by a single input parameter.

By analyzing the test data achieved from DSCS and NeoKylin, we achieve the following graphs which demonstrate the number of parameters responsible for each failure. Figure 6(a) and Figure 6(c) focus on the percentage of failures caused by a specific number of parameter(s) in DSCS and NeoKylin, respectively, while Figure 6(b) and Figure 6(d) illustrate the cumulative percentage of failures triggered by no more than a specific number of parameters. For example, in Figure 6(a), 47.8% (11 out of 23) of the failures in DSCS are caused by one input parameter solely, while 30.4% (7 out of 23) by the combinations of two parameters. In addition, as in Figure 6(b), 78.2% failures are caused by no more than two input parameters, while the percentage of failures caused by three parameters or less is 91.2%.

**Observation I**

*Similar to previous studies, we observe that a large percentage (more than 80%) of failures are triggered by no more than three input parameters.*

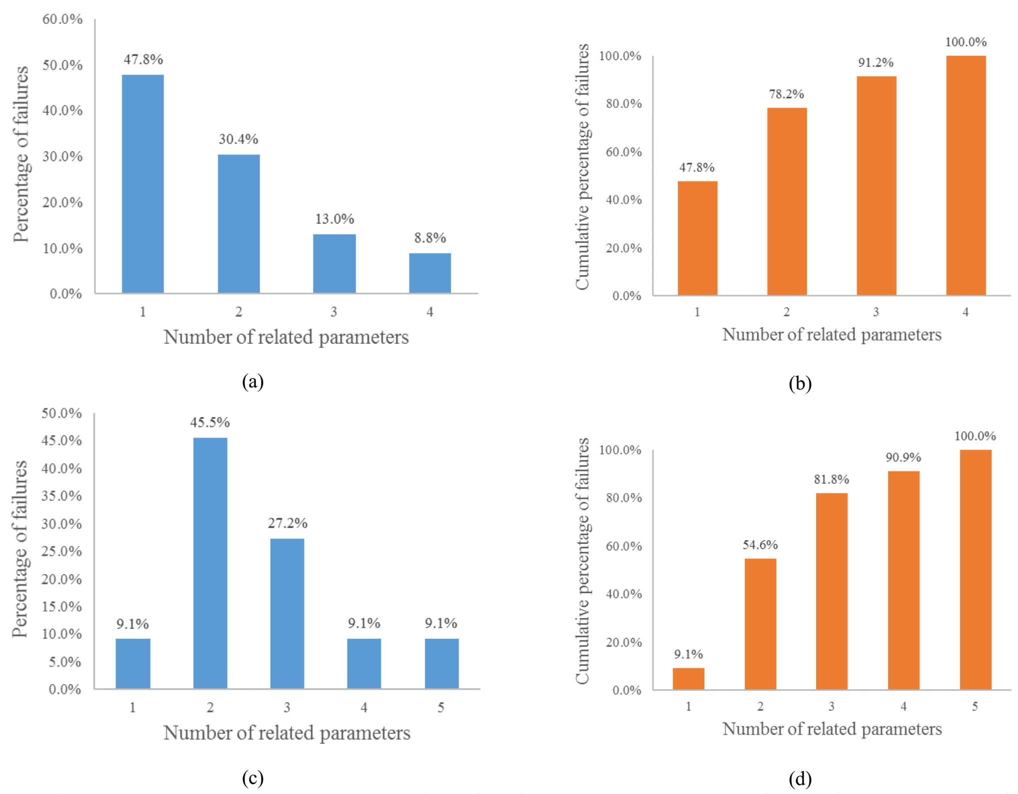


Figure 6. Number of failure causing parameter(s). (a) Percentage with fixed number of parameters for DSCS, (b) Cumulative percentage with no more than a fixed number of parameters for DSCS, (c) Percentage with fixed number of parameters for NeoKylin, (d) Cumulative percentage with no more than a fixed number of parameters for NeoKylin

### B. *What CT brings?*

Based on our discussion in Section IV, it is conceivable to reach the conclusion that test cases generated by CT could help us detect more bugs than traditional functional coverage-based testing. This is especially important when we are dealing with safety-critical software systems, for example the DSCS system.

Another significant positive result is that using CT reduces the staff costs needed to thoroughly examine the SUTs. Engineers at CEPREI are impressed that CT could vastly increase their efficiency in designing test cases by using the ACTS tool. In Figure 7, we include the average effort distribution of the two groups focusing on DSCS.

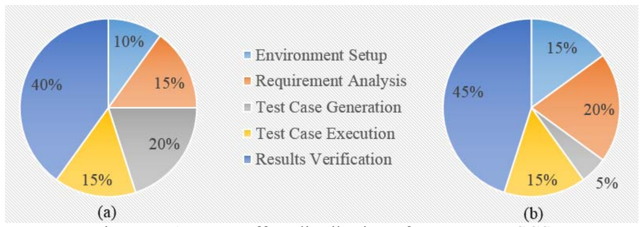


Figure 7. Average effort distribution of groups on DSCS
(a) FT (b) CT

As illustrated in Figure 7, by using CT, only 5% of the time costs will be spent on test case generation. In contrast, the percentage is 20% if using FT. As a result, with the help of ACTS which helps automate the process of test generation, CT saves a great deal of time needed to design test cases.

**Observation II**

*By applying CT, test effectiveness and efficiency can be significantly improved.*

### C. *Replacement vs Supplement*

Engineers at CEPREI are impressed with the positive results of the three pilot projects. One question arises undoubtedly: can CT replace traditional functional coverage-based testing completely?

One may argue that since CT has significantly improved fault detection strength and reduced staff costs and time to design test cases with the help of advanced tools, CT can act as the primary quality assurance method in the future. However, this may not be the optimal solution.

Though CT is effective and efficient, engineers have difficulty finding which parameter(s) actually cause(s) a specific failure. Due to the fact that there exist a certain portion of faults triggered by exactly one input parameter, it becomes considerably harder to figure out which parameter in the failed test cases actually triggers the bug. As a result, engineers need to go through all the input parameters provided by a specific failed test case in order to understand why the system enters the incorrect state, wasting a lot of time.

Therefore, we believe the optimal solution is to combine traditional functional coverage-based testing and CT in order to establish a stronger quality assurance framework. Traditional functional coverage-based testing will be first carried out based on the engineers' perception of the system and their experiences. Some easy-to-detect bugs can be discovered. Meanwhile, the understanding of the SUT will be strengthened, and engineers will know how to better design the test cases using CT. Once the functional coverage-based testing is done, CT can be applied to help detect more complicated bugs which are triggered by multiple input parameters.

Another interesting phenomenon is that by applying CT, more than 80% of the bugs can be discovered by executing less than 20% of test cases. For example, with respect to testing the DP III of DSCS, six of the seven bugs can be discovered by the first 20 test cases executed. Similar or even the same bugs occur continuously when executing the rest of test cases. Hence, in practice, we may not need to execute all the test cases generated by CT. A portion of test cases, for example 20%, can be executed first instead. Engineers will then fix the bugs encountered. After these bugs are fixed, we re-execute all the test cases to eliminate the remaining bugs. In this way, we not only reduce the noise during the test execution but also have a chance to validate the bugs encountered have been successfully resolved.

**Observation III**

*The optimal choice is to combine CT with other techniques to form a more robust quality assurance process.*

## D. *Culture Change*

Since most of companies have fairly rigid testing process, it is always a big challenge introducing a new technique to the members of testing teams. Two side-effects arised during the application of CT to the three pilot projects.

*1) Unrealistic expectations* sometimes have negative impacts on where and how a new tehcnique can be applied in real-life industrial projects. For example, when testing the NeoKylin operating system, the engineers believe that by applying CT, not only can more bugs be detected but also the size of test set becomes smaller. However, this may not be possible since the quality of the original test set needs to be improved with additional test cases. As a result, engineers may find CT ineffective, which could hinder the application of CT in future projects.

*2) Change of testing process* will also be a great challenge. In addition to the fact that engineers have to learn the basic concepts of CT, our investigation illustrates that CT should be incorporated into the early stage of testing in order to show the optimal effects. However, this requires the change of the entire testing process within the whole testing group. On the other hand, appropriate training as well as necessary documents and materials should be provided to ensure each engineer properly understands the procedure of a new technique. For large companies such as CEPREI, it can be extremely difficult with the pressure from various aspects (e.g. deadlines, inadequate training, insufficient staff with proper understanding of the technique, and so on).

**Observation IV**

*Obstacles exist in the application of new technology.*

## E. *Improvements in Supporting Tools*

By applying the ACTS tools to the three pilot projects, we also find that the tool itself can be further improved.

*1) Prioritization of Parameters.* Engineers at CEPREI argue that the parameters should be prioritized so that more combinations among those more "suspicious" input parameters can be generated for those parameters which are critical to the SUT. In other words, higher order of CT can be applied to specific parameters while test generation for other parameters still follows the lower order. For example, with respect to a program P with five input parameters, namely A, B, C, D, and E, we would like to generate test cases with different strengths. Assume engineers of the testing group believe that based on their previous experiences, A and B will be more likely to trigger faults. As a result, 3-way CT should be applied for A and B, while 2-way CT for the rest of input parameters.

*2) Concept of Sequence.* During the application of CT to NeoKylin, which is an operating system for desktop computers, the input parameters of the system are different operations performed by the user. An important factor that should be considered is to include the sequence of operations. For example, in order to complete a job $X$, two operations, namely A and B, should be performed. However, there is no explicit order of the two operations. As a result, the system performs as expected if operation A is executed first. However, if we execute operation B before A, a particular fault occurs and renders the system in an incorrect state. The phenomenon cannot be ignored when a relative large number of operations are performed sequentially. Therefore, the concept of sequence should be introduced and realized in the supporting tools of CT.

**Observation V**

*Improvements should be implemented with respect to the supporting tools of CT.*

## F. *Universal or Specialized?*

Based on our investigation of the three pilot projects, we are confident that CT can be applied to various types of software systems. However, the application is easier for some systems yet relatively harder for others.

Of the three pilot projects, DSCS, which is an embedded software system, is a perfect platform for the application of CT. Since the values of inputs are represented by one or several bits of a string, the inputs of the system can be easily parameterized, which greatly mitigates the obstacles in designing test cases using CT. However, for the NeoKylin operating system, the parameterization of inputs of the system is relatively difficult. Various operations should be identified as well as the values associated with an operation. As a result, more staff costs are spent on designing test cases for NeoKylin compared to those for DSCS.

**Observation VI**

*Though applicable for most of software systems, the application of CT could be easier for specific types of systems.*

## G. Lack of Proper Training Materials

It will be very difficult for a new technique to be accepted and adopted by industry without a set of comprehensible and practical training materials. Nowadays, most of the materials introducing CT are published papers and reports. However, engineers find those materials quite difficult to follow due to their technical details. Moreover, engineers are more interested in how to apply CT to different types of systems, which could act as important references for their on-going projects. However, to the best of our knowledge, we can't find such documents in the existing literature.

**Observation VII**

*Proper training materials are needed.*

## VI. Conclusion and Future Work

Our efforts to apply CT in real-life industrial settings have been demonstrated to be successful, which not only improves the effectiveness of the test sets but also reduces staff costs in the testing process. Though some concerns still exist, engineers at CEPREI are enthusiastic in applying CT to other projects.

In the next phase, our first step, in cooperation with CEPREI, is to apply CT to more projects with wider scope to further illustrate the applicability of the technique. Next, improvements for the CT tools will be implemented, which include both the prioritization of parameters and the concept of sequence. In addition, more detailed training materials will be compiled which concentrates more on the practical instructions of CT and the experiences retrieved from past projects.

## References

1 ACTS, a combinatorial test genertion tool, http://csrc.nist.gov/groups/SNS/acts/

2 J. Bozic, B. Garn, I. Kapsalis, D. E. Simos, Severin Winkler and F. Wotawa, "Attack Pattern-Based Combinatorial Testing with Constraints for Web Security Testing," in *Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security* (QRS), pp. 207-212, Vancouver, Canada, August 2015.

3 K. Burr and W. Young, "Combinatorial Test Technique: Table-based Automation, Test Generation and Code Coverage," in *Proceedings of the International Conference on Software Testing, Analysis, and Review* (STAR), San Diego, 1998.

4 G. Dhadyalla, N. Kumari, and T. Snell, "Combinatorial Testing for an Automotive Hybrid Electric Vehicle Control System: A Case Study," in *Proceedings of the Seventh IEEE International Conference on Software, Testing, Verification and Validation Workshops* (ICST Workshops), pp. 51-57, 2014.

5 M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the In-Paramter-Order Strategy for Constructing Covering Arrays," *Journal of Research of the National Institute of Standards and Technology*, vol. 113, no. 5, September 2008.

6 L. S. Ghandehari, J. Czerwonka, Y. Lei, S. Shafiee, R. Kacker, and D. R. Kuhn, "An Empirical Comparison of Combinatorial and Random Testing," in *Proceedings of the Seventh IEEE International Conference on Software, Testing, Verification and Validation Workshops* (ICST Workshops), pp. 68-77, March 31 to April 4, Cleveland, Ohio, 2014.

7 J. D. Hagar, D. R. Kuhn, and R. N. Kacker, and T. L. Wissink, "Introducing Combinatorial Testing in a Large Organization: Pilot Project Experience Report," in *Proceedings of the Seventh IEEE International Conference on Software, Testing, Verification and Validation Workshops* (ICST Workshops), pp. 153, 2014.

8 J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker, "Introducing Combinatorial Testing in a Large Organization," *IEEE Computer*, vol. 48, no. 4, pp. 64-72, April 2015.

9 D. Hillmer, "Introducing Combinatorial Testing in the Organization: a Report on a First Attempt," in Proceedings of the Eighth IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 1-9, April 2015.

10 Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: Efficient Test Generation for Multi-way Combinatorial Testing," *Software Testing, Verification and Reliability*, vol. 18, pp. 125-148, 2008.

11 D. R. Kuhn and R. N. Kacker, "Practical Combinatorial (t-way) Methods for Detecting Complex Faults in Regression Testing," in *Proceedings of the IEEE 27th International Conference on Software Maintenance* (ICSM), pp. 599, September, 2011.

12 D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial Coverage Measurement Concepts and Applications," in *Proceedings of the Sixth IEEE International Conference on Software, Testing, Verification and Validation Workshops* (ICST Workshops), pp. 352-361, 2013.

13 D. R. Kuhn, R. Kacker, Y. Lei, and J. Hunter, "Combinatorial Software Testing," *IEEE Computer*, vol. 42, no. 8, pp. 94-96, 2009.

14 D. R. Kuhn, R. N. Kacker, Y. Lei, and J. Torres-Jimenez, "Equivalence Class Verification and Oracle-free Testing using Two-Layer Covering Arrays," in *Proceedings of the Eighth IEEE International Conference on Software, Testing, Verification and Validation Workshops* (ICST Workshops), pp. 1-4, 2015.

15 D. R. Kuhn, R. C. Bryce, F. Duan, L. S. G. Ghandehari, Y. Lei, and R. N. Kacker, "Combinatorial Testing: Theory and Practice," *Advances in Computers*, vol. 99, pp. 1-66, 2015.

16 D. R. Kuhn, D. R. Wallace, and A. M. Gallo Jr, "Software Fault Interactions and Implications for Software Testing," IEEE Transactions on *Software Engineering*, vol. 30, no. 6, pp.418-421, 2004.

17 M. Palacios, J. Garcia-Fanjul, J. Tuya, and G. Spanoudakis, "Automatic Test Case Generation for WS-Agreements using Combinatorial Testing," *Computer Standards & Interfaces*, vol. 38, pp. 84-100, 2015.

18 NIST Report, "Software Errors Cost U.S. Economy $59.5 Billion Annually", NIST Planning Report 02-3, May 2002J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

19 L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Combinatorial Test Generation for Software Product Lines Using Minimum Invalid Tuples," in Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE), pp. 65-72, 2014.