An Intelligent Approach to Verification and Testing of the Configurator

Pei-Lei Tu†, Jen-Yao Chung‡ and Christos N. Nikolaou‡

†IBM Enterprise Systems
Poughkeepsie, NY 12602

‡IBM Thomas J. Watson Research Center, P.O. Box 704
Yorktown Heights, NY 10598

Abstract

The configurator verification requires three basic processes: generating the test data, analyzing the actual and expected outputs, and fixing the deviations. The typical verification processes of today's configurators are to manually provide the test case and expected output, and then analyze the differences between the expected and actual test outputs. This manual approach not only constrains the testing and verification of the configurator but also is not feasible for the large computer configurator program such as IBM ES/9000™. An intelligent approach to verification and testing of large computer configurators is introduced where test data are generated automatically and test results are analyzed intelligently with little human intervention. This approach utilizes a generic configurator model where the a priori computer configurator knowledge is captured and applied to generate a large number of potential test data and to analyze the test results automatically. With this intelligent approach, human experts' knowledge and skills can be better utilized to initialize the configurator model and to review the potential faults of the configurator program as revealed by the testing results.

1. Introduction

Verification and testing of a computer configurator is different from the typical hardware and software testing. A hardware testing is complicated due to numerous testing conditions for a large number of electronic components but there usually exist some "gold units" that can be used as a basis to determine if the test results are accurate. The testing of the software program is complicated since the implementation and the logic flow of each software program is very flexible. But for software programs such as arithmetic comparisons and functional subroutines, the expected outputs are predictable so the accuracy of the testing outputs can be checked automatically. Traditionally, verifying the accuracy of a computer configurator program requires a product expert providing the test case and its expected output, comparing the expected and actual outputs, and analyzing the discrepancy. This configurator verification process is as complicated as the typical hardware and software testing processes since a large computer usually has numerous components, multiple options for each component, and enormous combinations of components. The expected output of a configurator should be a valid customer configuration which is projected based on individual customer's requirements. As opposed to the hardware and software testing where the expected outputs might be already known, the expected test outputs of the configurator are not easily predictable but are projected on each test input. Therefore, the verification and testing of the configurator are usually done manually.

Currently, as far as we know, most computer configurators operate with manual verification process with limited testing capabilities. Some configurator verification might add structured design to the manual testing process [3] but the entire manual testing process is still very time-consuming. In general, this manual configurator verification not only constrains the testing scope of the configurator but also requires a lot of manpower and development cycle time for the

™ Trademark or registered trademark of the International Business Machines Corporation.
implementation of a good quality configurator. For larger computers such as the IBM ES/9000™ family where the product has more components and more combinations of configurations than smaller computers, such a manual verification and testing approach is usually not feasible. The issue of how to provide automation or intelligence to test data generation and analysis for the large computer configurator has become increasingly important in business competitive edges where the production cost of the configurator can be reduced and the quality of the configurator can be improved. In this paper, we will present an intelligent approach to verify a large computer configurator for IBM ES/9000™ with a large number of test cases generated automatically and the test results analyzed intelligently with little human intervention.

The rest of the paper is organized as follows. Section 2 states the various testing strategies of configurator verification. The intelligent approach of automating configurator verification is outlined in Section 3. Sections 4, 5 and 6 discuss the details of this approach for test data generation, output analysis and structured expert review. Conclusions are made in Section 7.

2. Verification and Testing of Configurators

There are several verification and testing strategies for a large software program using varying amount of test data: exhaustive testing, selective testing, formal proofs, and partial verification with automated evaluation systems [2]. We will analyze each testing strategy and suggest the best one for verification and testing of large configurator programs.

It is not always possible to have an exhaustive testing of any software program because of the size of all permuted testing cases. In the configurator domain, all possible combinations of customer configuration increase exponentially as the number of features increase from a minicomputer to a mainframe computer. For an ES/9000™ machine, the number of all configuration permutations could be millions. Even if all configuration permutations are used as test data, it will be extremely human-intensive and usually impossible to verify and analyze the test outputs.

Selective testing focuses on the selected part of the configurator program with specified testing inputs. This is the most common and easiest testing strategy which the product expert determines specifically what should be tested. Since the selected test data are limited, the test results usually can be analyzed manually. A verification tool will only need to provide the user with panels for specifications of the test case and the expected output, and the tool can generate discrepancies between the expected and actual configuration outputs for users' review. With its limited testing capability, the selective testing should not be the sole testing strategy but should be used in combination with other testing approaches for the purpose of flexibly selecting a small portion of the configurator for regression test or repetitive test. In practice, selective testing has been commonly used as the only testing strategy for configurator verification.

Besides the insufficiency of testing a large scale software program such as the ES/9000™ configurator, this testing strategy can be easily biased by the user.

Using the formal proof to verify and test a large software program is usually costly and the required advanced mathematical skills are not widely applicable. The software technologies for the implementation of large configurator programs might not be the same for every computer product and they are usually not algorithmic. Based on these, this testing strategy, formal proof, should not be considered for the verification and testing of configurator.

Partial verification with automated verification tools is to provide automated tools for software verification to an acceptable degree of reliability and performance. Studies [2] indicated that existing automated verification tools had reduced the software program errors drastically with several basic functions: (1) predicting the ripple effects of program modifications; (2) generating and evaluating test cases for a thorough and systematic program testing; (3) generating a database for future maintenance. This testing strategy has outperformed the above three testing strategies in many software applications. We'll call this testing strategy a comprehensive testing since comprehensive, instead of exhaustive or selected, amount of test data are used. Even though this testing strategy has proved to be a better testing strategy than the others for a large software program, this testing strategy has never been used in any configurator verification. Two major requirements are that (1) the verification tool needs to generate sufficient test data to disclose every potential fault of the configurator program but not with the exhaustive test data; (2) the verification tool should be capable of predicting and analyzing the test results.
The best verification tool for large configurator programs would need to satisfy these two requirements and the above basic functions of comprehensive testing. The objective of this paper is to present an intelligent approach of design and implementation of such a verification tool. We will also demonstrate that the size of the exhaustive test data can be drastically reduced but its testing scope stays the same. In addition, the tool should also accept user-selected test data, predict and analyze the test results for the user. This is to provide the selective testing of ripple effects of “what-if” conditions (e.g., adding or deleting a feature), and for divide-and-conquer to focus on a particular component of the large computer.

3. Design of Intelligent Configurator Verification

The configurator verification requires three basic processes: generating the test data, analyzing the actual and expected outputs, and fixing the deviations. The typical verification processes of today’s configurators are to manually provide the test case and expected output, and then analyze the differences between the expected and actual test outputs. These manual processes as shown in Figure 1 require a lot of human intervention and are error-prone. The test data are usually limited and are not sufficient for a large configurator verification.

To provide sufficient and comprehensive testing of large configurator programs, the automated verification tool needs to either automate or structure the above processes to facilitate large scale testing. The intelligent

---

**Figure 1. Typical Configurator Verification Processes**
configurator verification we present consists of automatic test data generation, intelligent output analysis and structured expert review as shown in Figure 2. Two types of test data selection are provided in this intelligent verification: comprehensive testing and selective testing. If the comprehensive testing is selected (e.g., testing several models of the product), all required test data will be created automatically based on a comprehensive test data generation algorithm. The expected test outputs will be generated by utilizing a configurator model where the a priori domain knowledge is captured. The discrepancies between expected and actual test results will be analyzed intelligently and the further analyses will be provided for domain experts' review. Test data will be classified as GOOD or BAD, and will be stored in the test database for future repetitive or regression tests. Each of the three processes is essential to configurator verification and cannot exist without the others. Even if there is an automatic test case generator to generate as many test cases as possible, the entire verification process will be very human-intensive and impractical to analyze enormous outputs manually. Without a structured expert review process, there won't be effective and efficient verifications for later regression testing or "what-if" analyses where slight variations of previous test data are used. Each of these three processes for automated configurator verification is elaborated in the next three sections where the characteristics of configurator knowledge as defined below are used.

3.1. Characteristics of Configurator Knowledge

![Figure 2. The Architecture of the Intelligent Configurator Verification](image)
In general, selection of a computer product requires specifications of the model (e.g., ES/9000™ model ##), and a collection of marketable features (e.g., central storage, extended storage, channels, etc.) with each feature characterized by its options, i.e., the choices of quantity for a feature. For example, the central storage might have several options: 256 megabytes, 512 megabytes or 1024 megabytes, but only one of these options can be chosen for the central storage of a configuration. A feature's options could vary depending on the model selected. Usually high-end models will offer more central storage than low-end models.

A computer configuration is characterized by: (1) the selected model and features; (2) valid combinations of features and the model. The former are the individual selections of models and features. The latter are from the affiliations between the model and features. Conceptually, these configuration characteristics can be represented in a model/feature association diagram as shown in Figure 3 and this diagram should be generic enough to represent most of the IBM configurators. The diagram exhibits a configuration with the selected model and features, represented as circles, and also the associations existed among features as affected by the model, represented as rectangles.

Suppose $m$, $f$, $o$ and $r$ denote a model, a feature, an option and a relation, respectively. A configuration $config$ consists of the selected model and a set of component list $cl$, i.e., $config = (m, cl)$ where $cl = \{(f(i), o_{m,f}(j))\} i = 1, 2, \ldots, t\}$ and each of the $t$ components is described by the product feature and its selected option. Relations of features can be categorized exclusively into four distinct classes: features are pre-requisite or co-requisite, features are mutually exclusive, and features have a total quantity constraint, i.e., $R$, a set of $r = (R_{pre-req}, R_{co-exist}, R_{mut-excl}, R_{tot-quant})$. For simplicity, let's assume only binary relations existed between features, these four types of relations can be represented as follows:

$$R_{(m, \text{pre-req})} = \{(f_i, o_j) \in cl \text{ if } (f_i, o_j) \in cl\}$$

where feature $f_i$ is a pre-requisite of feature $f_j$;

$$R_{(m, \text{co-exist})} = \{(f_k, o_j) \in cl \text{ and } (f_k, o_j) \in cl\}$$

where features $f_k$ and $f_j$ should co-exist;

---

Figure 3. Model/feature Association Diagram
where features $f_i$ and $f_k$ are mutually exclusive;

$$R_{m, mutu - excla} = \{((f_i, o_j), (f_k, o_j)) \mid$$

if $(f_i, o_j) \in \text{cl}$ then $(f_k, o_j) \notin \text{cl}$ or

if $(f_k, o_j) \in \text{cl}$ then $(f_i, o_j) \notin \text{cl})$$

where features $f_i$, ..., $f_k$ have a total-quantity constraint and $op$ is the total-quantity operator. The above relations of features in a model could exhibit two extreme cases such as,

1. “the simplest case with $R_{m_0} = \{((f, o))\}$ where $f$ could be any feature”

Under this circumstance, each feature is totally independent of the other features. The configurator will only need to examine the selection of each individual feature or model without concern for connections or constraints existed between features.

2. “the most complicated case with $R_{m_0} = \{((f, o), ..., (f, o))\)”

This is the most complicated case where each feature is related to each other. Whenever a feature is selected, the configurator has to check if this selection satisfies this feature's relationship to every previously selected feature.

In practice, most computer configurators fall in between these two extreme cases. Most of the feature relations in IBM ES/9000™ are binary with a few exceptions of ternary relations.

The above representation can be used to represent most of the configurator knowledge. The completeness of this representation depends on the completeness of its elements: models, features, features' options and relations.

4. Automatic Test Data Generation

It's not difficult to design an automatic test case generator to simulate all possible combinations of customer orders. In fact, the issue of an automatic test data generator is how to generate a reasonable size of test cases which are sufficient for an acceptable and reliable verification of configurator. What's also important is how can the test results be analyzed automatically so the entire test data generation and verification process can be automated with little human cooperation. We will address the automatic test data generation in this section and discuss how can the test results be analyzed intelligently in the next section.

An exhaustive testing is a brute-force approach where the search space for test data is extremely large but the test data generation is straightforward. A comprehensive testing with reduced search space for test data is not straight-forward and usually requires heuristics. The essence of the heuristic algorithm we use is from the previous defined configurator knowledge. Test data generated from this heuristic algorithm will be able to disclose any potential fault of the configurator program. Closely examining the configurator knowledge, we know that if features have no relations at all, test data of their exhaustive combinations are irrelevant since these test data are always valid but cannot explore any error of the configurator. These test data can be eliminated and the size of test data can be reduced drastically but the results of configurator verification are not affected. Details of this heuristic algorithm are described below.

4.1. Comprehensive Test Data Generation Algorithm

**Comprehensive Test Data Generation Algorithm**

1. Identify all the models in the set $M$.
2. Collect all the product features and conceptual features to be included in the configuration.
3. Identify all the relations $R$ between features.
4. Decompose features into subfeatures, if necessary, to reflect every relation (dependent, mutually exclusive or total quantity constraint). This collection of features or subfeatures will be the feature set $F = \{f_i \mid i = 1, ..., n\}$ used to generate test data.
5. Identify the options for each feature, i.e., set $O_{f_i} = \{o_j \mid f_i \in F\}$. The set $O_{f_i}$ collects all the feasible options for the feature $f_i$ independent of the model. The purpose is to uncover the common error of choosing the unallowable option for any feature $f_i$ in any model.
6. To generate the component list of each test case, do the following for each model $m$ in $M$:
   a. Starting with one relation in $R$ of features $f_i$ and $f_k$, use the features' option sets $O_{f_i}$ and $O_{f_k}$ to generate the exhaustive combinations of the
component list \((f, o), (f', o'))\), and randomly choose a fixed option for each of the rest of the features in \(F\) (e.g., the first option).

b. Similarly, for each of the rest of the relations, generate the exhaustive combinations of options for its features and the options for the rest of the features stay the same.

c. Eliminate any duplicate data generated from Steps a and b.

d. For every feature not in any relation, generate its exhaustive options and the options for the rest of the features stay the same.

Using the above heuristic algorithm, we can detect every potential fault in the specifications or implementations of configurator program. We can also eliminate a huge number of test data in the exhaustive testing since they are irrelevant to the configurator verification. With this heuristic algorithm, we can automatically generate test cases for comprehensive testing with drastically reduced size of test data but acceptable verification. The completeness of test data following this heuristic approach depends on the completeness of models, features, options and relations.

Details of this heuristic algorithm will be illustrated in the following example. For simplicity, assume there is only one model but five features \(f_1, f_2, f_3, f_4\) and \(f_5\) with options \((01, 01), (02, 02), (03), (04, 04)\) and \((05, 05)\), respectively. Features \(f_1\) and \(f_4\), \(f_2\) and \(f_3\) are related, either pre-requisite, co-requisite or mutually exclusive to each other. The automatically generated comprehensive test data following the above heuristic algorithm are shown in Table 1. The heuristic algorithm won’t generate test data of other exhaustive combinations of features, such as \(f_1\) and \(f_5\), \(f_1\) and \(f_2\), \(f_1\) and \(f_3\), \(f_1\) and \(f_4\), \(f_2\) and \(f_3\), since they are irrelevant to the configurator and cannot detect any error of the configurator. A total of 22 test data can be eliminated but the testing scope of the configurator is as comprehensive and sufficient as the exhaustive testing.

4.2. Automatic Test Data Generation for an IBM ES/9000™ Configurator

For example, each model of an IBM ES/9000™ processor offers the following basic features [1]:
- Central Storage (CS)
- Expanded Storage (ES)
- Channels, Parallel or Serial
- Vectors

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>(f_1)</th>
<th>(f_2)</th>
<th>(f_3)</th>
<th>(f_4)</th>
<th>(f_5)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>011</td>
<td>021</td>
<td>031</td>
<td>041</td>
<td>051</td>
<td>Generate the combinations of (f_1) and (f_4)</td>
</tr>
<tr>
<td>2</td>
<td>011</td>
<td>021</td>
<td>031</td>
<td>042</td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>021</td>
<td>031</td>
<td>043</td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>012</td>
<td>021</td>
<td>031</td>
<td>041</td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>012</td>
<td>021</td>
<td>031</td>
<td>042</td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>012</td>
<td>021</td>
<td>031</td>
<td>043</td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>011</td>
<td>021</td>
<td>031</td>
<td>041</td>
<td>051</td>
<td>Generate the combinations of (f_2) and (f_5)</td>
</tr>
<tr>
<td>8</td>
<td>011</td>
<td>021</td>
<td>031</td>
<td>041</td>
<td>052</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>022</td>
<td>031</td>
<td>041</td>
<td>051</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>011</td>
<td>022</td>
<td>031</td>
<td>041</td>
<td>052</td>
<td></td>
</tr>
</tbody>
</table>
- Integrated Cryptographic™ (ICRF™)
- High Performance Parallel Interface™ (HiPPiTm)
- Machine colors
- Console Color
- Sysplex Timer Attachment™
- Transaction Processing Facility Enabler™ (TPF™)
- Power Unit
- ESCON™ Analyzer
- Processor Controller
- Special Handling Tools

Besides the above product features, one conceptual feature, the customer location, will be considered since the customer's locations can affect the customer's selection of the power unit and the special handling tool. The entire collection of product features or conceptual features of an ES/9000™ processor are shown in Figure 4.

Based on all the identified relationships between the above features, we need to decompose several features since they can be installed on both sides of a multiple-processor machine and are distinguished in the machine configuration. These include features CS, ES, Vectors, ICRF, HiPPi, Sys-Timer-Attach, and TPF. They are decomposed into subfeatures CS-A and CS-B, ES-A and ES-B, Vector-A and Vector-B, ICRF-A and ICRF-B, HiPPi-A and HiPPi-B, Sys-Timer-Attach-A and Sys-Timer-Attach-B, TPF-A and TPF-B, respectively. One other feature which needs further decomposition is Channel since there are two types of channels, parallel

---

Figure 4. ES/9000™ Processors

™ Trademarks or registered trademarks of the International Business Machines Corporation.
and ESCON, and they are distinguished in the configuration. The feature decomposition and feature relationships are shown in Figure 5.

Once all the models, features, options and relations are identified, the comprehensive test data generation algorithm can be used to generate test data automatically. This example also demonstrates a comparative result between an exhaustive testing and the heuristic comprehensive testing. The total number of test cases form the exhaustive testing for this example is 2,381,847,700,000 while the heuristic algorithm generates a total of 3,126 test cases. Even though the size of test data has been drastically reduced but the configurator verification is not affected.

5. Intelligent Output Analysis

We already showed a comprehensive testing algorithm for automatic generation of data. If the test results of these data require human analyses, it is not only error-prone but also infeasible for mainframe computer configurators such as IBM ES/9000™ where even the reduced size of comprehensive test data is numerous. In order to provide a total automation to the verification and testing of the configurator, we need to automate both processes: test data generation and output analysis.

The automated output analyses require the verification tool to be able to analyze the test results in the same way as a human expert does. The tool will need to predict the expected configuration output and analyze this output with the actual configurator output. If both

![Diagram](image-url)
outputs are the same, i.e., the test case is a valid configuration or it is invalid with the same diagnosed violations, the configurator program has been tested successfully on this particular test case. However, if there are discrepancies between the expected and actual outputs, further investigations are needed to determine the causes. We have utilized the a priori configurator knowledge to generate test data automatically. Here we will demonstrate the intelligent verification tool for automatic output analysis by using the previously defined configurator knowledge.

Given a test case of configuration config = (m, cl), where cl = \{(f(i), a_{m,f}(j)) | i = 1, 2, ..., l\}, the validity of this test case is determined by the accurate selections of individual model and feature (e.g., the feature is offered in the model) and the correct combinations of features (e.g., two features are not mutually exclusive). First, the accuracy of the selection of individual model and feature is equivalent to the validity of each component (f(i), a_{m,f}(j)) in the configuration. Based on our previously proposed philosophy of test data generation, test data should exhibit any incorrect selection of a feature's option, such as a low-end model is ordered with too much central storage or a high-end model is ordered with too few channels. This is the reason we use the set O_{i,m} where O_{i,m} = \{a_{i,m} | f_j \in F\} in the test data generation algorithm. But to verify the accuracy of the component in the test data, we need to use the set O_{m,n}, where O_{m,n} = \{a_{m,n} | f_j \in F\}. The set O_{i,m} includes all the possible options for feature \( f_j \) disregarding which model is selected while the set O_{m,n} includes all the possible options for feature \( f_j \) when the model \( m \) is selected. Second, the test data is examined to see if the feature has the right option as selected, such as (model##, \{(CS-A,128MB), (CS-B,256MB), (ES-A,0MB), (ES-B,0MB), (Parallel-Channels,32), (ESCON-Channels,32), (Vector-A,1), (Vector-B,1), (ICRF-A,1), (ICRF-B,1), etc.\}). First, each component is examined to see if the feature has the right option since the chosen option could be offered by the feature but is not allowed in the selected model. Second, each component is checked against every relation in the relation set R. A summary of this process is shown in Table 2. The first analysis, Min/Max satisfied?, is to check the validity of the selected option for each feature. The following analyses, Pre-req correct?, Co-exist correct?, Mutually exclusive correct?, Total quantity satisfied are to examine each component against every relation in the relation set R. This test case is found to violate one mutually exclusive relation between features Vector and ICRF. If the configurator indicates that this is a valid configuration, it means the configuration program is inaccurate and is not implemented with the mutually exclusive relation between features Vector and ICRF. But if the test result from the configurator is consistent with the findings from the verification tool, it means the configurator program is accurate on this particular test case.

This intelligent output analysis can be repeated for a large number of test data in a batch process. Since the test data are generated by navigating through the configurator model and the model/feature association diagram, test data can be presented in a structured sequence and be grouped based on their variance such as different options for two features in a relation. If there is a missing implementation of the mutually exclusive relation between features Vector and ICRF in the configurator program, this error will be a common error to a group of test data grouped based on this relation. Once this error is identified, this error only needs to be corrected once. The user can then choose to re-run only this group of test data to ensure that a common error existed in a group of test data is fixed.

6. Structured Expert Review

If the automated output analysis shows the same result as the actual result from the configurator program, a satisfactory test case will be classified and collected into the GOOD test database, and an unsatisfactory test case is
Table 2. An Example of Automated Test Analysis of ES/9000™ MODEL

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(CS-A, 128MB)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(CS-B, 256MB)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(ES-A, 0MB)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(ES-B, 0MB)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(Parallel-Channels, 32)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(ESCON-Channels, 32)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(Vector-A, 1)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Vector-A should be mutu. exclus. with ICRF-A</td>
</tr>
<tr>
<td>(Vector_B, 1)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Vector-B should be mutu. exclus. with ICRF-B</td>
</tr>
<tr>
<td>(ICRF-A, 1)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>ICRF-A should be mutu. exclus. with Vector-A</td>
</tr>
<tr>
<td>(ICRF-B, 1)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>ICRF-B should be mutu. exclus. with Vector-B</td>
</tr>
<tr>
<td>(HiPPI-A, Yes)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(HiPPI-B, Yes)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(Machine-Color, BLUE)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(Console-Color, BLUE)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(Sys-Timer-Attach-A, Yes)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(Sys-Timer-Attach-B, Yes)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(TPF-Enabler-A, Yes)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>(TPF-Enabler-B, Yes)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

case will be classified and collected into the BAD test database. If there is any discrepancy between the expected and actual results, a human product expert will need to determine the fix to the configurator program and re-run the test data after the configurator program is fixed. In every software verification and testing, it is very important to keep both GOOD and BAD test cases in each test iteration for the following purposes: (1) serves as the base for later testing and (2) provides continuity or tracking for later testing since a bad test case might become a good one and a good test case could become a bad one if the product specifications change.

With the automated process for test data generation and output analysis, a human product expert doesn't have to create test data and manually analyze the results. Instead, the expert's knowledge and skills can be more effectively and efficiently utilized to review the analysis results generated automatically. This approach not only
enables a large scale verification and testing of configurator programs but also facilitates the repetitive regression tests.

7. Conclusions

In this article, we address the issue of automating the verification and testing of a large computer configurator program. In contrast to the current manual approach with limited testing scope, we present an intelligent approach to generate test data and analyze the test results automatically. The selection of test data is not the same as the exhaustive testing where every possible combination of features is generated. Instead, the number of test data is drastically reduced but the testing scope of the configurator verification is not affected. Both the test data generation and the output analysis are done by fully utilizing the configurator domain knowledge which is systematically represented as a configurator model. The completeness of this configurator model directly affects the quality of test data selection and the accuracy of output analysis. With this intelligent approach, human experts' knowledge and skills can be better utilized to initialize the configurator model and review test results already analyzed, instead of providing test data and expected output which are very time-consuming and error-prone for a large configurator program.

References

