2014 IEEE International Conference on Software Testing, Verification, and Validation

System Level Combinatorial Testing in Practice – The Concurrent Maintenance Case Study

Paul Wojciak
IBM Systems & Technology Group
2455 South Road
Poughkeepsie, NY 12601, USA
wojciak@us.ibm.com

Rachel Tzoref-Brill
IBM, Haifa Research Lab
Haifa University Campus
Haifa, 31905, Israel
rachelt@il.ibm.com

***Abstract*—Combinatorial test design (CTD) is an effective test design technique that reveals faults resulting from parameter interactions in a system. CTD requires a test space definition in the form of a set of parameters, their respective values, and restrictions on the value combinations. Though CTD is considered an industry best practice, there is only a small body of work on the practical application of CTD to industrial systems, and some key elements of the CTD application process are under-explored. Specifically, little consideration has been given to the process for identifying the parameters of the test space and their interrelations, how to validate the test space definition remains an open question, the application of CTD in reported work concentrates mostly on function or interface level testing and is hardly expanded to other levels of testing, and there is a significant lack of evaluation of the degree to which CTD helped improve the quality of the system under test.**

In this work, we analyze the continuous application of CTD in system test of two large industrial systems: IBM® POWER7® and IBM® System z®. For POWER7, CTD was used to design test cases for server concurrent maintenance. The application of CTD was in direct response to inconsistent reliability of those features on the prior POWER6® servers, and resulted in noteworthy quality improvements on POWER7. Success with POWER7 led to application of the methods to System z Enhanced Driver Maintenance testing, also featured in this work. To the best of our knowledge, this is the first comprehensive analysis of CTD usage in system test of industrial products, and the first analysis of long term use of CTD.

We describe the methodology that we followed to define the combinatorial test space, while answering some unique challenges rising from the use of CTD to design functional test cases at system level rather than at interface or function level. We also describe our methodology for evaluating the test space definition and continuously improving it over time. In addition, we describe advanced CTD features that we found helpful for achieving an effective yet affordable test plan. Finally, we quantitatively and qualitatively evaluate the overall effectiveness of CTD usage, and show that it resulted in significantly improved server concurrent maintenance features.

I. Introduction

Combinatorial test design (CTD), a.k.a. combinatorial testing, is an effective test design technique for coping with the verification challenge of increasingly complex software systems. As the size of test spaces continuously grows, exhaustive verification methods become impractical. Functional testing, on the other hand, is prone to omissions, as it always involves a selection of what to test from a possibly enormous test space.

CTD addresses this challenge by a systematic selection of tests, which is based on the observation that in most cases, the appearance of a bug depends on the interaction between a small number of features, or parameters, of the system under test (SUT). Experiments show that a test set that covers all possible pairs of parameter values can typically detect 50% to 75% of the bugs in a program [7], [22]. Other experimental work has shown that typically 100% of bugs can be revealed by covering the interaction of between 4 to 6 parameters [15]. Hence, in CTD, the test space is manually modeled by a set of parameters, their respective values, and restrictions on the value combinations, termed a *combinatorial model*. A valid test in the test space is defined by an assignment of one value to each parameter without violating the restrictions. A subset of the space is automatically constructed so that it covers all valid value combinations (a.k.a interactions) of every t parameters, where t is usually a user input. The most common application of CTD is known as pairwise testing, in which the interaction of every pair of parameters must be covered, but, in general, one can require different levels of interaction for different subsets of parameters. Each test in the result of CTD represents a high level test, or a test scenario, that needs to be translated to a concrete executable test.

Existing studies and reports [3], [24], [9], [5], [7] indicate that CTD is very effective for a variety of system types and testing domains, and is considered best practice when the tested functionality depends on multiple factors such as inputs, configuration elements and data items. However, there is only a small body of work on the practical application of CTD to industrial systems. A recent survey [17] by Nie et al. reveals that only around 5% of the publications on CTD explore the process for identifying the parameters of the test space and their interrelations, which is a crucial process since it determines to a large extent the effectiveness and efficiency of the resulting test plan. Nie et al. indicate that how to effectively and efficiently model the test space for CTD, as well as how to validate the model, remain open questions in this field. Nie et al. also indicate that there is a lack of empirical results in CTD, and there is a need to conduct more studies and collect more evidence to fully understand the limitations and strengths of CTD. Another observation is that in the vast majority of the publications on the practical application of CTD, the test space

978-0-7695-5185-2/14 $31.00 © 2014 IEEE
DOI 10.1109/ICST.2014.23

parameters represent either input parameters at the interface or function level or configuration variables, whereas using CTD to design functional test cases at the system level is hardly explored. Nie et al. call for the expansion of CTD to other levels of testing beyond function test.

In this work, we describe the long term use of CTD to design test cases at system level for features of two large industrial systems: IBM® POWER7® concurrent maintenance features (CHARM), and System z® Enhanced Driver Maintenance (EDM). These features, collectively referred to as concurrent maintenance features, perform changes, additions, and repair of the system hardware and code while the system is running live business applications. Performing live maintenance rather than scheduling an outage for maintenance is extremely important because it helps preserve the constant availability of the server. However, it also constitutes a very complex technical challenge, both design-wise and test-wise. It is important to note that while these features manipulate hardware systems, they are in fact implemented in code, thus the functionality under test is software (firmware) based. Previous experience with the concurrent maintenance features on POWER6® revealed some significant deficiencies in system test, and therefore CTD was used to design the system level test cases for POWER7 during the past three years. Following the success of CTD on CHARM and the observation that some quality concerns of EDM had a direct connection to parameter interactions, CTD was also applied to system test of EDM, and we report on this effort as well. To the best of our knowledge, this is the first comprehensive analysis of CTD usage in system test of industrial products, and the first analysis of long term use of CTD.

Identifying the test space parameters and values at the system level is much more challenging than in the traditional CTD application at the function level or for optimization of configurations, since in the latter, the input parameters and configuration variables can be used as a starting point for the test space definition, whereas in the former, one needs to identify the test space from scratch. We suggest a methodology for identifying parameters and values at system level and describe its application to our case studies.

We also suggest a methodology for validating the resulting test space definition. It is based on the analysis of test results from CTD in each test cycle as well as analysis of field results, in order to evaluate the test space definition and improve it for the next test cycle. We used this methodology to continuously evaluate and improve the CHARM models during the past three years.

Another contribution of this work is a quantitative and qualitative evaluation of the effectiveness of CTD usage on CHARM, that shows that the use of CTD resulted in markedly improved concurrent maintenance features – field results show a dramatic increase in the success rate of the concurrent maintenance operations, and an elimination of the crashes for all but a handful of cases. Furthermore, most of the remaining failures are due to factors on which testing has no impact. According to [17], the evaluation of the quality of the SUT following the application of CTD is yet another area which is considerably under-reported in published studies on CTD – there was only one such study out of 93 surveyed papers.

In addition, we describe various advanced CTD features that we used and found beneficial. These features are provided by our CTD tool, IBM Functional Coverage Unified Solution (IBM FOCUS) [12], and helped us through the test space modeling and test plan refinement to reach an effective yet affordable test plan.

To summarize, the contribution of this work is as follows: we report and analyze two large industrial case studies for the long term application of CTD. We describe a non-traditional application of CTD at system level, and suggest methodologies for defining a test space and for validating it. We demonstrate the use of advanced CTD features to improve the resulting test plan. Finally, we provide an evaluation of the effectiveness of CTD and the degree to which it helped improve the quality of the features under test.

In the following we describe in Section II the concurrent maintenance features, as well as the previous system test approach that was in use and its deficiencies. Section III presents the methodology we followed in applying CTD, including the process for defining the test space and for validating it, and the refinement of the resulting combinatorial test plan. In Section IV we provide an evaluation of the effectiveness of CTD, that shows how CTD significantly improved the quality of the concurrent maintenance features, followed by a discussion in Section V of the lessons learned from our experience. Section VI explores related work on the practical application of CTD. Finally, Section VII summaries our work and provides future work directions.

II. The Systems Under Test

In this section we describe the concurrent maintenance features under test, as well as the previous test approach taken and its deficiencies. We start with some background that puts these features in context and highlights their importance to customers.

A. CHARM: IBM® Power® Server Concurrent Maintenance

Enterprise computing environments require constant availability. Time lost to unavailable server resources can cost financial institutions millions of dollars [1]. Lost time can come from scheduled and unscheduled outages. Unscheduled outages can originate from events outside the server, from failing hardware components within the server, or software problems associated with the server. Scheduled outages represent the largest contributor to enterprise system unavailability and lost time [23]. Routine maintenance for system hardware upgrades, code updates, or repairs are the biggest contributor to scheduled outages. Given that enterprise servers rarely experience significant lulls in their utilization, adding more capacity to those systems while they are running is very valuable to the customer. However, adding more physical processor, memory or Input/Output (I/O) hardware while the server is running is a technically complex undertaking akin to adding more cylinders

to your car while driving on the highway. Scheduled outages of enterprise servers for the purpose of upgrading or repairing hardware can nearly be eliminated by concurrent maintenance features. IBM's Power enterprise class servers address this with Central Electronic Complex (CEC) Hot Add Repair and Maintenance (CHARM) [4].

CHARM firmware enables Power systems to avoid scheduled outages for capacity and maintenance operations. Built from interconnected compute nodes, high end Power servers support adding, changing, and repairing processor, memory and I/O capacity. These actions are performed while the server may be hosting over a thousand virtual server instances each running business of scientific applications.

POWER6 CHARM test design employed a combination of functional and structural test cases [13]. Functional test cases were derived from the design specification. Structural tests were based on similar functions on the IBM system z class of enterprise servers. A system test in CHARM, a.k.a `a trial`, is concerned with performing the maintenance operations end to end including the full sequence of steps, i.e., several firmware components being tested together and in sequence. With this design approach, the POWER6 test was thought to be a success. However, field results were not as anticipated. Field concerns had several causes. Limited servicer training and failing hardware components were out of test's scope. However, firmware problems and some design concerns had escaped test. Root cause analysis showed attributes influencing failures had not been part of the test plan. Secondary errors during concurrent maintenance plagued the field but played little role in the test plan. Server configurations at the customer often differed considerably from those of the system test machines. Firmware state dependence also intersected with failing cases. As it turned out, there was much omitted from the test plan.

The detailed evaluation of the POWER6 concurrent maintenance experience led to a number of test changes for POWER7 server concurrent maintenance testing, the most significant of which was test planning using the IBM FOCUS tool [12] that implements combinatorial test design. In Section III we go into detail on how this technology was used to improve the quality of POWER7 CHARM system test.

B. EDM: System z Enhanced Driver Maintenance

Enterprise customers demand and expect 24×7 availability from System z servers. Meeting that demand requires concurrency of maintenance including software, firmware and hardware. A major part of that equation is concurrent update of firmware fixes and delivery of new functions.

Comprising well over 10 million lines of code across 70+ major subsystems, System z firmware is the fundamental underpinning of the mainframe platform after the hardware itself. Firmware manages all aspects of the System z platform from day to day operations of the virtualized compute environment to recovery of errors occurring throughout the system. Firmware assures system availability demands are met for the mainframe customer. Maintenance and updates to System z firmware are concurrent including updating the entire firmware stack to a new version.

Known as Enhanced Driver Maintenance (EDM), this System z firmware capability describes concurrently updating the firmware stack, called the driver, from one version to another. Thousands of different mainframe operating system instances are permitted to be running customer production business applications and workloads during such an EDM update of the firmware driver. EDM must complete with no visible disruption to the running operating system environments. The vast majority of System z customers rely upon EDM when they upgrade to a new driver.

Test design for EDM had been based on functional tests against the design specification and structural tests based on the firmware implementation for over a decade. Verification of EDM on System z is the responsibility of the system test team. A single system test trial for EDM can take well over eight hours when setup, execution, and post update verification actions are considered. The trials themselves are very complex with many sub steps, each with definite sequences necessary to test the z firmware. As a result, getting the most return from every test trial is essential. Note that the system trials for both CHARM and EDM are performed by a human and contain almost no automation.

The resulting customer experience with EDM met expectations for multiple generations of mainframe. However, for the zEnterprise® z196 server family, customer experience with EDM fell short of expectations. While a very high success rate of over 97% was experienced in the field, the expectations were to have zero impacts. A thorough review of field results and factors contributing to EDM issues was conducted. Key findings were:

- Interaction of EDM firmware with non-EDM system functions and features correlated with issues
- Sequence of non-EDM system operations occurring before and after EDM correlated with issues
- A definite list of system functions interacting with EDM began to appear
- New functions in the updated firmware level merited particular attention (issues using a new function only when that function was concurrently activated)

Knowing that combinatorial test design (CTD) serves well for test designs sensitive to interactions, sequences, and different inputs, suggested a possible application for the next major system test of EDM on System z. Section III details this effort.

III. The CTD application process

As mentioned in Section II, the most significant change in the system test approach for POWER7 was in the test case design phase. Simply running more tests was not the answer. The tests needed to consider the configuration aspects contributing to failures, the secondary failure modes, and also the new features being added to POWER7 that might interact with concurrent maintenance. The number of test cases required to meet these needs appeared to be unaffordable considering the available time and manpower. The IBM FOCUS

tool [12] that implements CTD was applied to the challenge of creating a high quality yet affordable set of tests. The CTD models for CHARM cover configurations, secondary errors, and new features across all the POWER7 concurrent maintenance functions. Separate models were created for high end (IBM Power 795) 8 node machines and for mid-range (IBM Power 770 and 780) 4 node machines. In addition, the models evolved throughout three years and several different releases. Overall there were 12 different model versions in use. The final model for mid-range machines contained 10 parameters and 20 restrictions, and defined a test space of 109,776 valid trials. The final model for high end machines contained 9 parameters and 18 restrictions, and defined a test space of 13,944 valid trials.

EDM also experienced some quality concerns. As described in Section II, the concurrent upgrade failures were in combination with a previous or a following customer operations, indicating that interactions were important for triggering the failures. In system tests, however, there was no coordination between EDM and the operations happening before and after EDM. This observation together with the success of CTD on CHARM led to the application of CTD for system test of EDM on IBM zEnterprise EC12 [8]. Model parameters for EDM reflect the three major sequence steps in the firmware process: Pre-EDM, During-EDM, and Post-EDM. The parameter values are designed to introduce pairwise interactions amongst major System z firmware functions and EDM. The final version of the CTD model for EDM contained 10 parameters and 20 restrictions, and defined a test space of 2.25E+8 valid system trials.

Note that the models for CHARM and EDM contain a relatively small number of parameters. The challenge is to maintain a small and comprehensive model at a reasonable abstraction level for such complex systems, rather than letting the number of parameters and values explode by defining a too granular test space. In the following we describe the methodology we used for creating the above CTD models while coping with this challenge, as well as our methodology for validating the test space definition and improving it. In addition, we describe how we refined the resulting test plans using advanced CTD features, available in IBM FOCUS.

A. Defining the test space

The main step in the application of CTD is the definition of the test space in the form of parameters, their respective values, and restrictions on value combinations (a.k.a the combinatorial model). This crucial step provides the foundation for the following steps, and largely determines the quality of the resulting test plan. Any important points of variation overlooked by the model result in coverage omissions and reduced effectiveness of the test plan, and any redundant parameters or values in the model result in redundancy and reduced efficiency of the test plan.

1) Identifying the parameters: The first task in defining the combinatorial model is identifying the parameters of the model. Understanding the differences between function level software testing and system level software testing is important for grasping how the modeling parameters used also will need to differ. Function level testing is generally concerned with inputs and outputs of individual product features or software components. Such testing is concerned with verifying coverage of each input and output of the function, and variations are typically derived from the input and output value differences. The sequences and interactions of these input and output values are formally considered and the combinatorial modeling serves very well for this purpose.

System level software testing is most notable for the concentration on the overall stress to the system under test. Multiple applications operate simultaneously exercising a number of different functions at the same time. Where the function test would concentrate on each function or feature somewhat in isolation, the system test is concerned with interactions amongst different functions and features when they are all running together. As a result, the system test model would select parameters to represent the different functions and features to be exercised in concert.

System test often refers to different combinations of compute applications as a workload. For example, an on-line trading workload could consist of the web serving front end, an application serving middle tier, and a database back end. The model for system-testing such a system could choose parameter values to represent workload intensity (throughput, utilization), workload ramp time (increase throughput or utilization from low to high and back to low), workload composition (trade size, company or commodity, and market) and workload constraints and error conditions. A functional test model in this example would be more likely to validate each of the three tiers and their characteristics independently. At most, the functional test would model specific transaction flow from end to end.

In the case of CHARM, system test is concerned with performing the maintenance operations end to end including the full sequence of steps. The interactions modeled are between the firmware supporting CHARM and other software components executing on the system while maintenance is performed. As a further example, EDM system testing represents three distinct phases of operation with a set of preconditions occurring before the upgrade, a set of during conditions that happen while the firmware is managing the update, and a set of post conditions being done after the upgrade. The system test model treats the various functions, features, system stress, system states and errors that can occur as part of those three distinct EDM phases.

From our experience, to cope with the higher abstraction level that modeling at the system level requires, answering a series of questions often helps:

1) Does the system test have a flow with distinct steps? This characteristic helps the person doing the model derive parameters that reflect steps in the flow. CHARM has distinct operations with a set flow, and errors are introduced and detected at specific points in that flow. EDM has a notion of `pre-cdu`, `during-cdu`, and

post-cdu parameters. In both models, the parameters expose these flow steps to advantage.

2) Are there system states, usage patterns or user stories? can they be represented as categories of system states or use cases? This is beneficial in limiting parameter value explosion. For example, in CHARM, the Workload parameter represents a usage pattern. In EDM, the inclusion of various system operational stress levels was captured by the During_Workload parameter.
3) Does bad path need explicit testing? For example, we had bad path related parameters in the CHARM model called Error_during_CHARM, and State. In CTD, bad path requires special consideration. We further elaborate on modeling of bad path in Section III-A3.
4) What configuration characteristics are meaningful? This tends to be the traditional use of CTD for function test and it applies to system test as well. For example, node and failing_FRU reflect configuration aspects for CHARM. The sensitivity of EDM success to the starting firmware version was included through the From_Level parameter, which depicts allowed combinations of code from over 70 firmware components.
5) Are there orthogonal system operations or applications whose intersection with the primary function to be tested must be considered? For example, the EnergyScale parameter in CHARM models covers firmware functions that interact meaningfully with CHARM code.
6) Is there any field experience or other external sources from which important factors can be derived? For example, in the CTD model for EDM, error cases derived from field experience and RAS statistical models were included as parameters. Associations between Pre and Post conditions were also created in the model, as correlation was suggested by field experience. In addition, some functions and features were included in the model based on either their interaction with EDM as seen through field results or their being historically complex.
7) Are there any new additions to the system that need to be considered? For example, some functions and features were included in the model for EDM since they were new or recent additions to System z.

2) Assigning values to the parameters: For each parameter, one needs to decide what level of granularity of its values is needed. For example, in system test, the modeler can be less granular regarding the workload parameter, and consider its values in terms of bandwidth, throughput or capacity as subset or percentage of the maximum.

Usually in CTD, the granularity of each parameter is determined separately when it is defined and added to the model. However, we noticed that it is valuable to also consider the general granularity of the model. As parameters are added to the model, one should be able to verify that they all have similar levels of granularity. When a certain parameter has decidedly more granularity than all other parameters, one would see in the proposed tests that there is not much difference between many of them. The more granular parameter would force additional tests that are not meaningful. Some general well-known testing techniques such as equivalence partitioning and boundary value analysis [16] can be applied to help determine the granularity level of the parameter values [21].

In addition, when defining parameter values, considerations similar to the ones mentioned for parameter identification, such as field experience, complexity and novelty, can be applied.

3) Modeling bad path: Similarly to function test models, the system level test needs to be concerned with error handling by the software. Function test models might have granular error conditions aligned with each input to be modeled. The system test model would be more likely to include categories of errors from a number of functions and features perhaps basing them on the expected system response for each category. As can be seen, choosing modeling parameters for a system test requires a higher level of abstraction. CHARM models include the parameters Error_during_CHARM and State that represent the secondary error that may occur during CHARM and its associated firmware state, to achieve sufficient coverage and validate error handling related interactions. EDM models include Pre_Op_RAS, During_RAS, and Post_Op_RAS to address the need for error handling firmware coverage.

For CTD in general, it is especially important to specify the values and value combinations that represent bad path. Since each test in the CTD result may cover many unique combinations, if any of them represents bad path, the tests they appear in will exit prematurely or at least not execute the main path, and will not exercise the unique good path interactions they are supposed to cover. Avoiding such false coverage of required interactions is achieved by tool support for indicating what are the error conditions in the model [12], [6]. When bad path values are specified, each bad path test contains a single error condition, and the bad path tests aims only at covering interactions between bad path values and good path values. It is common that the logic of the error conditions does not depend on interactions, and so requiring level 1 interaction coverage for bad path is sufficient. However, in CHARM, the same error would often be handled differently depending on the firmware in control for the failing CHARM step or depending on the configuration of the system. As a result, covering interactions between bad path values and good path values was required. Since a lot of attention was given to bad path testing in CHARM where full recovery in the presence of errors was expected, the CTD test plan contained a ratio of up to 2:1 bad path versus good path tests.

4) Defining restrictions on value combinations: For both EDM and CHARM, restrictions were defined within the IBM FOCUS tool to eliminate invalid combinations of certain error conditions, configurations, and features. Restrictions can be easily specified within IBM FOCUS by viewing different projections of the test space on subsets of parameters, and excluding value combinations directly from the projection [20].

5) Determining the interaction coverage requirements: Different pairwise and t-wise model requirements were eval-

uated for our industrial applications. A pairwise requirements test approach was chosen after study of the 3-wise trial list and concluding the parameter interactions proposed seemed more than necessary. In addition, refinement of the pairwise requirements was done to remove insignificant interactions, as described in Section III-B.

B. Refining the CTD test plan

Once the test space is defined and a combinatorial test plan is generated, if the recommended list of tests is affordable, one can proceed with implementation of the tests. However, commonly it does not fit within the given testing resources, and there is a need to winnow the list of tests. One way to achieve reduction in the number of tests is by removing certain parameter combinations from the coverage requirements when they have no significant interaction. This should be done with care as not to lose important interactions, and requires tool support for mixed-strength test generation [20], [6].

Another way to achieve reduction is by handpicking an exclusion list from the resulting test plan. Careful consideration needs to be made on what to exclude from the test plan, since each test in the result of CTD may cover many unique combinations. The interactive refinement feature [19] in the IBM FOCUS tool was designed to support such cases. It allows educated decisions on what to exclude or modify in the test plan that results from CTD by displaying the coverage gaps that are introduced for each manual modification step. In addition to controlled reduction in the size of the test plan, this feature can be used to steer the test plan to focus more on higher importance areas, due to complexity or other considerations. This is achieved by requesting higher interaction coverage for certain value combinations of higher importance. We used the interactive refinement feature to reduce the appearance of lower importance values, while making sure that no coverage gaps are introduced in the process. For example, certain values of the `Error_during_CHARM` parameter of the CHARM models were handled this way since they represented lower importance failure types.

We also biased parameter value appearance in the recommended tests by assigning weights to parameter values [20], [6] as part of the input to the CTD algorithm. Weights are requested proportions of values in the test plan that results from CTD. Assigning a good set of weights often takes a subject matter expert with customer experience and field experience from prior products. Note that weights are considered as soft constraints by CTD tools, hence the resulting test plan might not fully satisfy the requested distribution.

C. Evaluating the combinatorial model

We evaluated the CHARM combinatorial models by analyzing the test results. We took advantage of the fact that system test was done in waves that each focused on specific concurrent maintenance tests. Wave objectives were comprised of new content and regression testing of existing content. Between the test waves, fixes were accumulated and delivered before starting the next system test wave. To evaluate the quality of the combinatorial model and refine it, we typically waited until the end of a test wave and reviewed the results from all the test trials. We then used the evaluation to guide the next test wave. The review considered the following questions:

- How was the overall defect discovery quality and rate?
- Did all parameters seem to matter?
- Can some parameters be eliminated?
- Are new parameters necessary?

We looked back on the experience where defects were found, the root cause of those defects, whether the root cause correlated in some observable way to the trial parameter values, and parameters where defects were not found. We tended to interpret the absence of any defect correlation with a parameter's entire set of values as an indication that parameter does not matter. However, we did not just take that parameter out of the combinatorial model yet. We waited until we had some amount of customer experience with the product and whether that evidence also indicated the parameter was irrelevant. For instance, the `ECO mode` parameter was in the models throughout all of POWER7 CHARM testing but no correlation with defects was found. Before eventually removing the parameter, its pairwise requirement was dropped. By removing the pairwise requirement, and by using weights, some coverage of ECO mode was retained. Another parameter from the POWER7 CHARM models, `EnergyScale`, did not correlate with any defects for the first two test waves. In the third test wave, a very significant defect discovery related directly to the `EnergyScale` parameter. As a result, the parameter was retained even though EnergyScale complicated test set up. Root cause of the EnergyScale issue indicated a missing system parameter, namely the workload or operational system stress during the test. This led to the introduction of the `Workload` parameter to new CHARM models.

Correlation of defect discovery to model parameters was employed to control the wave content and order of execution within a wave. Each test wave would try to diversify tests executed intending to expose certain interactions, by delaying tests that contained parameter values already correlated with an open defect in the current wave. For the regression part of the next wave, trials with parameter values previously finding defects were included, along with the new trials that were recommended by the IBM FOCUS tool.

IV. Evaluation

We present results from applying CTD to CHARM and EDM in two dimensions: improvement in test quality, based on analysis of system test results, and improvement in the quality of the server concurrent maintenance features, based on analysis of field results. We analyze field results for CHARM only. While we do not have field results yet for EDM, its preliminary evaluation on EC12 customer production machines has the best results ever seen for EDM to date.

A. Improvement in Test Quality

To evaluate test results, we use the well known test case effectiveness metric, which we refer to as the defects per trial

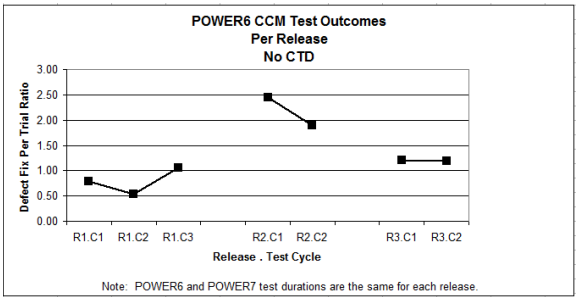

Fig. 1. POWER6 Test Outcomes Per Release

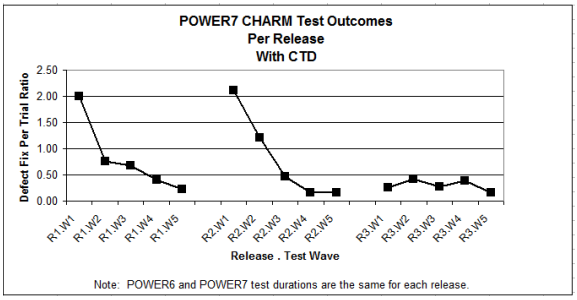

Fig. 2. POWER7 Test Outcomes Per Release

ratio. An important dimension to consider is how this ratio changes in the course of time, and whether any indications on product stability can be drawn from the nature of change along time. In addition, we qualitatively examine and analyze the nature of defects found.

First, we compare the defects per trial ratio for CHARM between POWER6 and POWER7. Figures 1 and 2 show the ratio obtained in the different test cycles of each release on POWER6 and POWER7, respectively. Note that POWER6 and POWER7 test durations are the same for each release.

As can be seen in Figure 1, on POWER6 the ratio never decreases below 0.5 and in fact is over 1.0 for the final test cycle of each release. This may suggest that the rate of defect discovery in the last test cycle can be an indicator of stability and release quality. Concluding that reaching a low defects per trial ratio in the last test cycle can foretell product quality is proven by the POWER7 ratio data. For each of the three releases, the final wave ratio is 0.3 or lower. Furthermore, the behavior of the defects per trial ratio on POWER7 perfectly correlates with CTD generated test trials. The first trials produced by CTD contain higher numbers of unique interactions. As a result, the defects per trial ratio of the first wave is high, around 2.0 or above. As the test waves progress, there are fewer interactions that remain to be tested in the CTD generated tests that were spread across the waves. New interactions are still uncovering new defects but the discovery rate drops significantly. This means that with CTD, more defects are found earlier during testing rather than being randomly distributed along time, and therefore stability can be reached faster. Notice that for POWER7 release 3 the

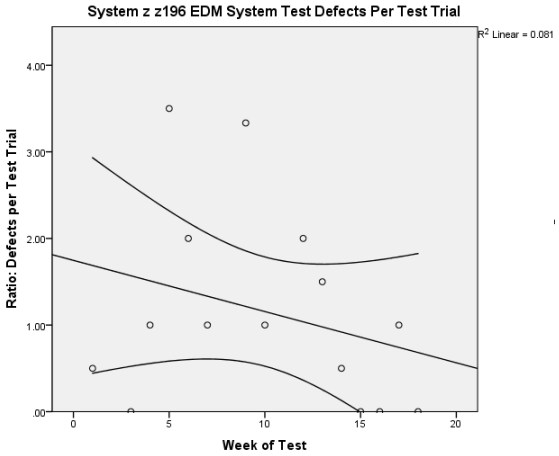

Fig. 3. System z z196 EDM System Test Defects Per Test Trail

downward slope was not as pronounced. This release included new CTD model parameters and some new parameter values. The `State` bad path parameter, the `Workload` parameter, and additional values for `Error_during_CHARM` produced new interactions that revealed defects. Trials were spread across waves. As it happens, this was done in such a way that unique interactions were postponed until later waves. This led to a similar number of defects per trial across all release 3 waves. By this point in time, the POWER7 CHARM base code was quite stable. This is seen in the overall lower ratio for all waves. Though less pronounced, decreasing defects per trial in the last wave did occur.

Next, we compare the defects per trial ratio for EDM between z196 and EC12, shown in Figures 3 and 4, respectively. Rather than showing the ratio per test cycle as we did for CHARM, we present the ratio per week. The reason is that as opposed to CHARM, for EDM there was only one long test cycle. For CHARM we do not present weekly defects per trial ratio since this data is unavailable.

The same trend that was observed for CHARM testing occurs also for EDM. The fit line and confidence interval using the least squares method allows one to observe the defects per test trial trend for both the z196 and EC12 EDM test efforts. There is a good fit for the EC12 test but not for z196. This is an indication that the EC12 EDM test was stabilized at the end of test. The z196 EDM test did not display such a trend. To further substantiate the claim that with CTD more defects are found earlier during test, bivariate correlation was performed for both the z196 and EC12 EDM tests using defects per trial ratio and week of test. The expectation was that the ratio should decrease as the weeks of test progressed. The EC12 test analysis showed that there was a significant negative Pearson correlation, with r=-0.746, n=17, p=0.001. Such a trend helps emphasize the quality of the EDM trials. Over the course of the test there are fewer defect revealing interactions remaining to be exposed. With the correct test plan, the product stabilizes

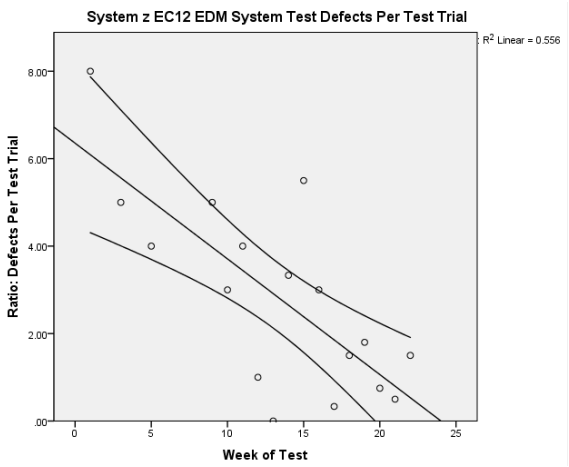

Fig. 4. System z EC12 EDM System Test Defects Per Test Trail

and quality is achieved. The bivariate analysis of the z196 EDM data revealed no monotonic relation between week of test and defects per trial ratio, with a Pearson correlation of -0.285 and a Spearman's rho correlation of -0.301. Neither is statistically significant. This highlights an incomplete test plan suffering through continued high defect discovery throughout test leading to insufficient product stability.

Finally, we examine the defects found in system test of CHARM and EDM and analyze their nature. For CHARM POWER7, the first observation is that all the defects that were found were new, rather than a rediscovery of known issues. This fact is impressive considering that there was not much new functionality on POWER7, meaning that CTD allowed to extract more defects from a semi-stable POWER6 base code. Another observation is that there is indeed correlation between the new tests that came about from CTD usage and the defects that were ultimately found and extracted from the product. Root cause analysis showed that the defects discovered during test were indeed triggered by the interactions that CTD tests were exercising, rather than "accidentally" discovered during test. This evidence strongly supports the reasoning of the CTD approach in general, as well as the quality of the combinatorial models in use for CHARM on POWER7.

For EDM on EC12, we first observed that over 75% of the trials that came out of CTD found defects. This is especially important considering the need to get the most out of each of the trials due to their lengthy execution time. We then analyzed the defects that were discovered and fixed both during test and during field execution on z196 and EC12, and categorized them based on the type of code they were associated with. z196 test interaction omissions became visible as there were 50% more code categories associated with EDM field fixes than from test fixes. The test trials that came out of CTD for the EC12 system test covered all code categories that required a fix on z196, plus twice as many other code categories. Defects were found during system test on EC12 for about two third of these code categories, including for almost all

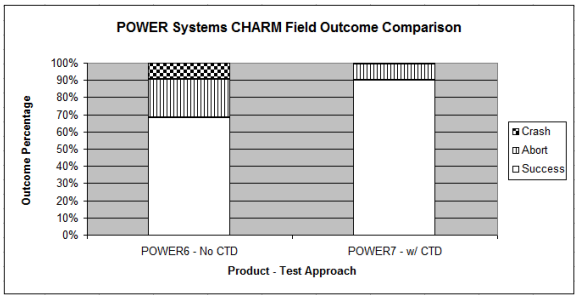

Fig. 5. Power Systems CHARM Field Outcome Comparison

categories that experienced failures in the field. This indicates that CTD helped close the coverage gaps experienced in the z196 test, as well as significantly improve the overall coverage.

B. Improvement in Field Quality

We compare field results of CHARM on POWER7 to those obtained on POWER6 in terms of the outcome of the concurrent maintenance operation. There are two types of possible failures of the maintenance operation, as witnessed on POWER6. The first is an abort, where the operation cannot be completed and there is a need to suspend it and retry at a later stage after consulting with the customer. The second and more severe failure type is a crash, where the entire machine comes down. The goal for POWER7 was to eliminate the crashes and absolutely minimize the aborts. As mentioned before, for EDM we do not have field results yet, however preliminary field evaluation is extremely positive.

As can be seen in Figure 5, dramatic quality improvement with CHARM has been experienced in the field for POWER7. While on POWER6 about 10% of the operations crashed, on POWER7 crashes were eliminated for all but a handful of cases. The percentage of aborts was reduced by half – from 20% to 10%, and respectively the success rate increased from approximately 70% to 90%.

Understanding why there remains a set of aborted CHARM operations on POWER7 requires further analysis. Figures 6 and 7 show the categories that are leading to the failures on POWER6 and POWER7 respectively, and the percentage of failures in each category. The `Firmware Error` category is a direct indicator of test quality, while the other categories point to factors outside testing control, ergo CTD's ability to influence. Note that we do not claim responsibility of CTD for failures that are the result of design limitations. The reason is that though design concerns can and in fact were discovered during testing, many of them were theoretical ones that could not be substantiated and fixed without field evidence.

Review of the categories shows that on POWER6, about one third of the failures derive from firmware errors, on which test has a direct impact. On POWER7 only 16% of the failures are still attributable to firmware, a dramatic improvement compared to POWER6, which leaves only little room for further improvement. By way of comparison, on POWER6 CHARM 10% of the overall operations failed due

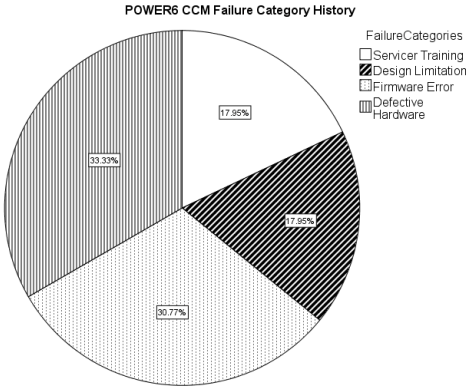

Fig. 6. POWER6 CHARM Failure Categories

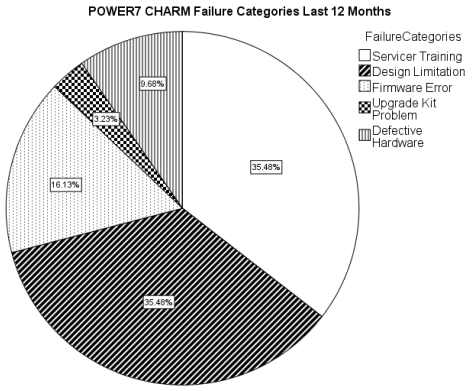

Fig. 7. POWER7 CHARM Failure Categories

to firmware errors, while on POWER7 CHARM only 1.6% of the overall operations failed due to firmware errors. Given the significantly higher success rate of POWER7 CHARM as well as the significantly lower rate of failures attributable to testing, we conclude that testing with CTD enabled reaching the "right" set of test cases.

V. Discussion

In the following we summarize the main lessons learned and the main points of value identified in retrospect of our practical experience with CTD throughout the past three years.

- **Closing coverage gaps.** CTD dramatically helped close coverage gaps. One way to achieve this was by guiding CTD towards covering the interactions pointed out by field failures, and as a result revealing also many more defects that were not yet experienced in the field.
- **Affordability.** In an industrial setting there are usually fixed budgets and schedules that determine what is an affordable test plan. CTD helped meet the affordability requirement with maximum coverage, as it allowed to easily generate alternative optimized test plans with different coverage levels, and choose between them. In addition, the need to perform manual modifications to the test plan is common in practice. Tool-assisted modification of the test plan is very useful and important to ensure that loss of interactions is minimal and controlled.
- **Level of abstraction.** The main challenge in deploying CTD at system test was to maintain the right level of abstraction when identifying the parameters and choosing their granularity. A model with too many parameters would not only be unaffordable and extremely hard to maintain, it would also overwhelm the human system tester with too many test parameters.
- **Importance of bad path.** It is very important to consider the bad path test space. At system level, the consequences of failures are much less predictable than at function level. Flexibility of the CTD tool in allowing to choose the interaction level of bad path testing is very useful.
- **Broader view of the test space.** CTD helped broaden the perceived view of the test space. Whereas in the past our focus was on the feature under test only, CTD forced us to think about the interactions surrounding the feature under test. Field experience suggested that was the case, but the structured CTD methodology raised the interactions to the surface. Furthermore, even if we knew the interactions that mattered, since there were so many of them, without CTD we could not have come up with an affordable and effective test plan, but rather with a very expensive one.
- **Importance of root cause analysis.** There was an ongoing real-time feedback loop involving the defect root causes, the combinatorial model, and the tests that have yet to be executed. This is especially valuable in an iterative testing process, where the impact of root cause analysis on further testing is immediate.
- **Test progress, completion and stability.** Our experience showed that the CTD methodology worked in practice – as fewer interactions were left to be tested, fewer defects were left to be found. It provided a better sense of test progress and completion. Furthermore, since the tests that came out of the CTD tool were front-loaded, i.e., the first tests contained more untested interactions than the later ones, more defects were found earlier during the test cycles, and as a result stability was reached faster.

One open question is how to verify that the test plan at hand is sufficient to achieve the required product quality outcome before any field results are available. This continues to be a limitation of system test that it seems even CTD cannot solve.

VI. Related Work

Existing studies on the application of CTD in practice concentrate mostly on function, interface or configuration level [14], [3], [24], [5], [7]. While we do not summarize this work here, we note that none of these studies evaluates the quality of the SUT following the application of CTD. The one study we found referring to field quality following the application of CTD is by Salem et al. [18], in which they developed a generic logistic regression model of predicting software failure based on the testing result of CTD.

We are aware of only two studies of CTD at system level. In [11], CTD is applied to design system level test cases for small, commercial satellite ground systems. While a reduction in the number of required test cases is shown, no quality evaluation is provided, nor a procedure for identifying the test space parameters. In [2], factorial design was used during system performance validation to give insight on which factors, or interactions between factors, affect the performance measure. Similarly to our test space evaluation process, factors were added and factors which had no effect were removed. Their work, however, did not contain an evaluation of the performance of the system in the field following the application of the proposed methodology, but rather evaluated only the reduction in effort required for understanding the performance influencing factors during system performance validation.

Guidelines for identifying the parameters and their values are reported in [7]. Based on experience from four case studies to which CTD at function or interface level was applied, the authors recommend to use an iterative process and expert knowledge to achieve a proper model of the test space. In [14], a method is proposed for determining the parameters by extracting them from the group of requirements in the scope of the function under test. Grindal and Offutt [10] present a basic eight-step process for input parameter modeling of two different types: interface-based and functionality-based. While providing some good modeling principles, their process is custom-designed for function and interface level and does not address the unique challenges of system level modeling.

Finally, we are unaware of any work that analyzes and evaluates the results of CTD along time.

VII. Summary and Future Work

In this work we present a comprehensive analysis of the use of CTD to design test cases at system level for features of two large industrial products: IBM POWER7 and IBM System z. Our study involves multiple under-explored aspects of CTD, including application of CTD at system level, long term use of CTD, the test space definition and evaluation processes, refinement of the resulting test plan to fit available resources, and evaluation of the quality of the features under test following the application of CTD. Our analysis of field results shows a dramatic improvement in the quality of the features under test, and that most of the remaining failures are due to factors on which testing has no impact.

In the future, we would like to further evaluate the results of CTD on EDM by analyzing full field results, once available. We would also like to analyze the few CHARM aborts on POWER7 resulting from firmware errors, understand why they escaped testing and how it could have been prevented.

Another direction we are exploring is the application of CTD at design time. Since CTD, and specifically the process for creating the test space definition, can uncover many design issues, it makes perfect sense to apply CTD at design time based on requirement documents. The obvious benefits are revealing design issues as early as possible in the development life cycle, and verifying the testability of the design.

Acknowledgment. The authors would like to thank Mike Duron and Butch McCardle for their exemplary leadership during the CHARM and EDM tests, respectively.

References

[1] A. Arnold. Assessing the financial impact of downtime (26th april 2010). IT-Director.com Website, http://www.it-director.com/business/costs/content.php?cid=12043. [Online; accessed 29-September-2013].

[2] T. Berling and P. Runeson. Efficient evaluation of multifactor dependent system performance using fractional factorial design. *IEEE Trans. Softw. Eng.*, 29(9):769–781, 2003.

[3] K. Burroughs, A. Jain, and R.L. Erickson. Improved quality of protocol testing through techniques of experimental design. In *IEEE Intl. Conf. on Record, Serving Humanity Through Communications*, volume 2, pages 745–752, 1994.

[4] CHARM. http://www-03.ibm.com/systems/power/hardware/whitepapers/770_780_cec.html. [Online; accessed 29-September-2013].

[5] M. B. Cohen, J. Snyder, and G. Rothermel. Testing across configurations: implications for combinatorial testing. *SIGSOFT Softw. Eng. Notes*, 31(6):1–9, 2006.

[6] J. Czerwonka. Pairwise Testing in Real World. In *Proc. 24th Pacific Northwest Software Quality Conf. (PNSQC'06)*, pages 419–430, 2006.

[7] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-Based Testing in Practice. In *Proc. 21st Intl. Conf. on Software Engineering (ICSE'99)*, pages 285–294. ACM, 1999.

[8] IBM zenterprise EC12. http://www-03.ibm.com/systems/z/hardware/zenterprise/zec12.html. [Online; accessed 29-September-2013].

[9] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Softw. Engg.*, 11(4):583–611, 2006.

[10] M. Grindal and J. Offutt. Input parameter modeling for combination strategies. In *Proc. of the 25th Conf. on IASTED Intl. Multi-Conf.: Software Engineering*, SE'07, pages 255–260, 2007.

[11] J. Huller. Reducing time to market with combinatorial design method testing. In *Intl. Council on Systems Engineering (INCOSE)*, 2000.

[12] IBM Functional Coverage Unified Solution (IBM FOCUS). http://researcher.watson.ibm.com/researcher/view_project.php?id=1871. [Online; accessed 29-September-2013].

[13] P. Jorgensen. *Software Testing A Craftsmans Approach*. Auerbach, 3nd edition, 2008.

[14] R. Krishnan, S. Murali Krishna, and P. Siva Nandhan. Combinatorial testing: learnings from our experience. *SIGSOFT Softw. Eng. Notes*, 32:1–8, 2007.

[15] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Engineering*, 30:418–421, 2004.

[16] G. Myers. *The Art of Software Testing*. ohn Wiley and Sons, 1979.

[17] C. Nie and H. Leung. A Survey of Combinatorial Testing. *ACM Comput. Surv.*, 43(2):11:1–11:29, 2011.

[18] A. M. Salem, K. Rekab, and J. A. Whittaker. Prediction of software failures through logistic regression. *Information Software Technology*, (12):781–789, 2004.

[19] I. Segall and R. Tzoref-Brill. Interactive refinement of combinatorial test plans. In *34th Intl. Conf. on Software Engineering (ICSE'012)*, pages 1371–1374, 2012.

[20] I. Segall, R. Tzoref-Brill, and E. Farchi. Using Binary Decision Diagrams for Combinatorial Test Design. In *Proc. 20th Intl. Symp. on Software Testing and Analysis (ISSTA'11)*, pages 254–264. ACM, 2011.

[21] I. Segall, R. Tzoref-Brill, and A. Zlotnick. Common patterns in combinatorial models. In *Proc. of the 2012 IEEE Fifth Intl. Conf. on Software Testing, Verification and Validation*, ICST '12, pages 624–629, 2012.

[22] K.C. Tai and Y. Lie. A Test Generation Strategy for Pairwise Testing. *IEEE Transactions on Software Engineering*, 28:109–111, 2002.

[23] P. Weygant. *Clusters for High Availability*. Prentice Hall, 2nd edition, 2005.

[24] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Proc. of the IFIP TC6/WG6.1 13th Intl. Conf. on Testing Communicating Systems: Tools and Techniques*, TestCom '00, pages 59–74, 2000.